A Beta Way: A Tutorial For Using Beta Regression in Pychological Research to Analyze Proportional and Percentage Data

Jason Geller[1], Robert Kubinec[2], Matti Vuorre[3], and Chelsea M. Parlett Pelleriti[4]

[1]Department of Psychology and Neuroscience, Boston College

[2]University of South Carolina

[3]Tilburg University

[4]School

Author Note

Jason Geller https://orcid.org/0000-0002-7459-4505

Robert Kubinec https://orcid.org/0000-0002-7459-4505

Matti Vuorre https://orcid.org/0000-0002-7459-4505

Chelsea M. Parlett Pelleriti https://orcid.org/0000-0001-9301-1398

Correspondence concerning this article should be addressed to Jason Geller, Department of Psychology and Neuroscience, Boston College, McGuinn 300, Chestnut Hill, MA 2467, USA, Email: drjasongeller@gmail.com

Abstract

Rates, percentages, and proportions are common in psychology and the social sciences, and usually treated as normally distributed. This practice, however, ignores salient features in the data, such as its natural limits at 0 and 1, and can thus lead to distorted estimates. Treating such outcomes as beta-distributed, instead, acknowledges these salient features and therefore leads to more accurate estimates. The adoption of such models remains limited, however. Our purpose is to guide researchers in using the beta distribution through illustrative analyses of a real example from the psychological literature. First, we introduce the beta distribution and the beta regression model highlighting crucial components and assumptions. Second, we show how to conduct a beta regression in R using an example dataset from the learning and memory literature. We then discuss extensions of the beta model, such as zero-inflated, zero- and one-inflated, and ordered beta models. To facilitate these analyses' more widespread adoption, we present a detailed code supplement at https://github.com/jgeller112/beta_regression_tutorial.

Keywords: beta regression, R, tutorial, psychology, learning and memory

A Beta Way: A Tutorial For Using Beta Regression in Pychological Research to Analyze Proportional and Percentage Data

In psychological research, it is common to measure performance (on a test or task), attitudes, and choices using outcomes expressed as proportions or percentages. To illustrate this in a practical context, consider a memory experiment in which participants read a short passage on a specific topic. After a brief distractor task, they complete a final memory test consisting of 10 short-answer questions, each assigned a different point value (e.g., question 1 might be worth 4 points, while question 2 might be worth 1 point). In this scenario, the primary focus is on the total number of points earned relative to the total possible points across all questions.

An important question arises: how do we analyze this type of data? In psychological research, proportional outcomes are frequently analyzed with methods that fall under the general or generalized linear (G/GLM) frameworks. General linear models assume normal distributions, such as t-tests and ANOVAs while generalized models assume non-normal distributions, with logistic regression being a common example.

One popular tool for estimation and hypothesis testing that most readers will be familiar with is ordinary least squares (OLS) regression. OLS regression assumes our response outcome is normal in nature. With our example above, we could calculate the proportion or percentage correct for each question (or overall performance) and submit it to a regression analysis. While this is a very popular approach in psychology and is used quite frequently (Sladekova & Field, 2024), OLS regression makes several key assumptions that make it not ideal for the data we have. These assumptions include (1) a linear and additive relationship between predictors and outcomes, (2) normally distributed errors with a mean of zero and constant variance, and (3) homoscedasticity. When these assumptions are met, OLS regression provides unbiased and reliable estimates. However, when dealing with proportional data, several of these assumptions are often violated due to their bounded nature (0 to 1) and tendency to exhibit non-constant variance (Ferrari & Cribari-Neto, 2004; Paolino, 2001). This makes OLS regression an unsuitable choice for such data.

Another popular tool for estimation and hypothesis testing with proportional data is logistic regression. In our example, we could model accuracy as the ratio of successes to failures—specifically, the number of correct responses relative to the total for each question. While this approach might seem appropriate, it presents several challenges when dealing with data near the boundaries of 0 and 1, which is a common characteristic of proportional or percentage outcomes in psychological research.

One major issue with this approach is boundary inflation, which refers to the inflated estimates that occur when data clusters too closely at the extremes of 0 or 1. Binomial models assume probabilities strictly between 0 and 1, and this assumption can lead to biased estimates, especially when a substantial proportion of responses cluster near the boundaries. Additionally, as proportions approach these extremes, variance decreases, resulting in overly narrow confidence intervals and an underestimation of variability. Real-world data often exhibit greater variability than what the binomial model can accommodate, leading to overdispersion that can distort statistical inferences. Moreover, the bounded nature of proportions complicates their interpretation within a binomial framework.

This discussion highlights the importance of selecting statistical models that accurately reflect the characteristics of your data. Choosing an inappropriate model can lead to misleading conclusions—for instance, failing to detect genuine effects (Type II errors) or falsely identifying effects that do not exist (Type I errors). To ensure valid and reliable inferences, researchers must carefully consider the structure of their data and the assumptions underlying their chosen models.

The challenges of analyzing proportional data are not new (see Bartlett, 1936). Fortunately, several alternative approaches address the limitations of commonly used models. One such approach is Beta regression, an extension of the generalized linear model that employs the Beta distribution (described in-depth below) (Ferrari & Cribari-Neto, 2004; Paolino, 2001). Beta regression offers a flexible and robust solution for modeling proportional data by accounting for boundary effects and overdispersion, making it a valuable alternative to traditional binomial models. This approach is particularly well-suited for psychological research because it can handle

both the bounded nature of proportional data and the non-constant variance often encountered in these datasets.A beginners guide to Beta regression

With the combination of open-source programming languages like R (R Core Team, 2024) and the great community of package developers, it is becoming trivial to run analyses like Beta regression. However, adoption of these methods, especially in psychology, is sparse. One reason for the lack of adaptation could be the lack of resources available to wider community (but see (Bendixen & Purzycki, 2023; Heiss, 2021; Vuorre, 2019), for excellent blog posts. This tutorial hopes to address this gap and gives researchers the tools and code needed to explore other statistical options in their research.

In this tutorial, we plan to (a) give a brief, non-technical overview of the principles underlying beta regression (keeping mathematics notation to a minimum), (b) walk-through an empirical example of applying beta regression in the popular R programming language and (c) highlight the the extensions which are most relevant to researchers in psychology (e.g., zero-inflated, zero-one-inflated, and ordered beta regressions).

To promote transparency and reproduciblity, the tutorial was written in R (v.4.4.2; R Core Team, [2024]) using Quarto (v.1.5.54), an open-source publishing system that allows for dynamic and static documents. This allows figures, tables, and text to be programmatically included directly in the manuscript, ensuring that all results are seamlessly integrated into the document. In addition, we use the `rix` (Rodrigues & Baumann, 2025) R package which harnesses the power of the `nix` (Dolstra & contributors, 2006) ecosystem to to help with computational reproducibility. Not only does this give us a snapshot of the packages used to create the current manuscript, but it also takes a snapshot of system dependencies used at run-time. This way reproducers can easily re-use the exact same environment by installing the `nix` package manager and using the included default.nix file to set up the right environment. The README file in the GitHub repository contains detailed information on how to set this up to reproduce the contents of the current manuscript

Beta regression can be estimated using various tools and approached from either a

frequentist or Bayesian perspective. We adopt a Bayesian framework primarily due to its practicality and the availability of advanced computational techniques ([Gelman et al. (2013); McElreath (2020); Johnson et al. (n.d.)). However, similar implementations are available within the frequentist framework (see supplemental materials).

In this paper, we demonstrate how to implement Beta regression models and their extensions using the brms (Bürkner, 2017) package. Additionally, we leverage the powerful `marginaleffects` package to extract meaningful estimates. We believe this combination provides an effective and efficient approach to performing beta regression, ensuring robust and interpretable results.

Beta distribution

Proportional data are bounded between 0 and 1 and often display heteroscedasticity. Common distributions that fall under the G/GLM often do a poor job of capturing this, necessitating the need to fit a different model to the data. For any statistical model, there are two main components: a random component that specifies the distribution of the response variable (such as a Poisson or binomial, which are part of the exponential family), a linear predictor that combines explanatory variables in a linear form, and a link function that connects the mean of the response variable to the linear predictor (Nelder & Wedderburn, 1972).

To deal with proportional data, we can fit a model with a Beta distribution (Ferrari & Cribari-Neto, 2004). The Beta distribution has some desirable characteristics that make it ideal for analyzing proportions: It is continuous, it is limited to numbers that fall between 0 and 1, and it highly flexible—it can take on a number of different distribution shapes. The shape which comprises the location, skew, and spread of the distribution are controlled by two parameters: *shape1* and *shape2*. `Shape 1` is sometimes called α and `shape 2` is sometimes called β. Together these two parameters shape the density curve of the distribution. To highlight this, let's suppose a participant got 4 out of 6 on a short answer question on a test. We can take the number of correct on that particular test item (4) and divide that by the number of correct (4) + number of incorrect (2) and plot the resulting density curve. Shape1 in this example would be 4 (number of points

received or successes). `Shape2` would be 2–the number of points not received (number of failures). Looking at Figure 1 (A) we see the distribution for one of our questions is shifted towards one indicating higher accuracy on the exam. If we reversed the values of the two parameters Figure 1 (B), we would get a distribution shifted towards 0, indicating a lower accuracy. By adjusting the values of two parameters, we can get a wide range of distributions (e.g., u-shaped, inverted u-shaped , normal, or uniform). In mathematical statistics, the beta distribution is known as the distribution for probabilities because probabilities are bounded but can never equal 0 or 1.

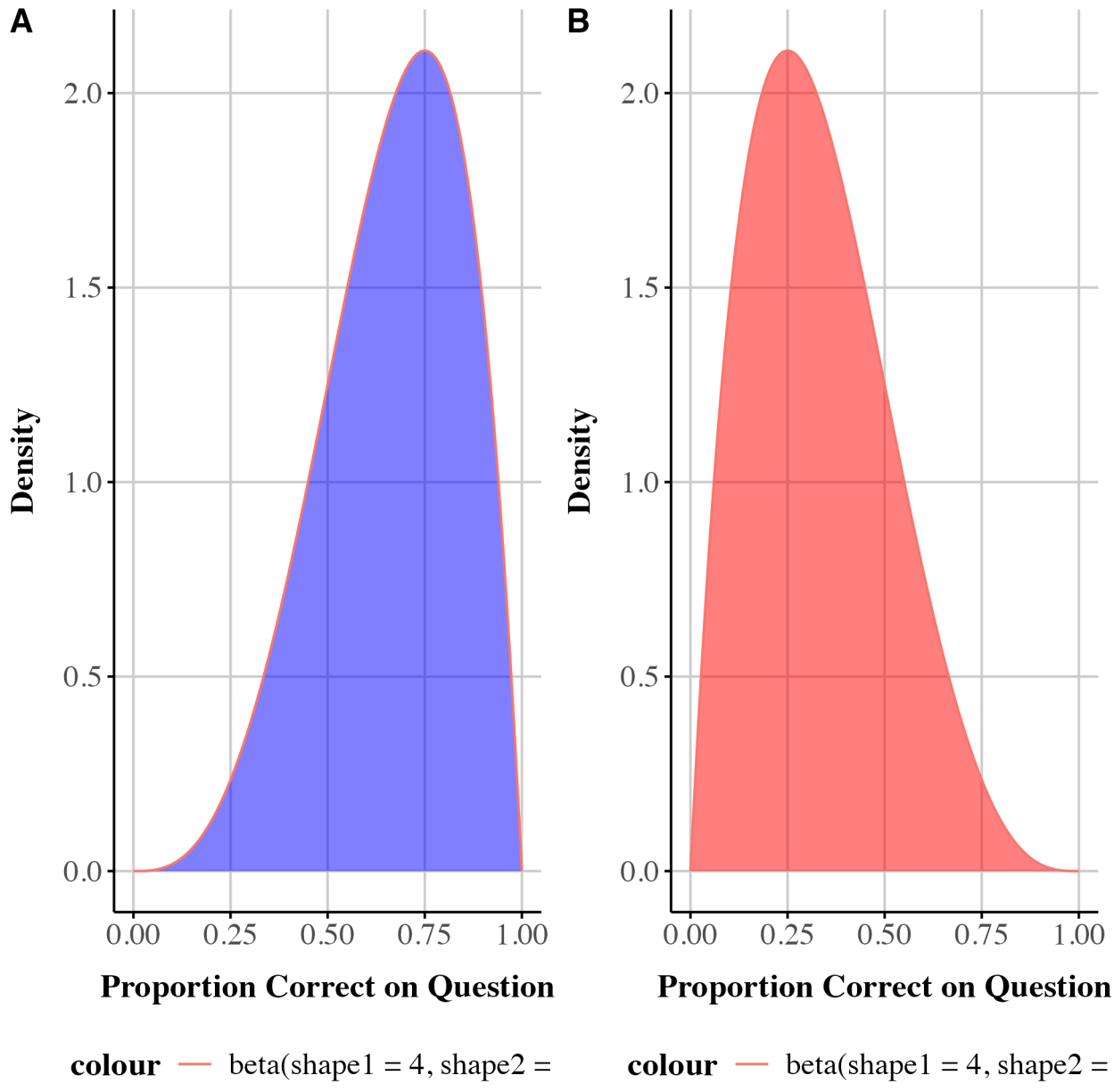
> **Note**
>
> Better example? Maybe more cog psych?

To make abundantly clear what the beta distribution is let's look at a political example—the 2024 election. For example, suppose we want to come up with our best guess of how Donald Trump might do in the 2024 presidential election given his performance in the 2020 election in terms of the number of electoral college votes he received (which are roughly proportional to the popular vote for president in the United States). In that case, we could use the beta distribution to get a sense of our uncertainty in that statement by plugging in 232 for the number of elector college votes Trump won in 2020 for α and 306 for the number of electoral college votes that Trump didn't win for β–i.e., all the electoral college votes Joe Biden won in 2020 :

```
# simulate 10000 samples from the beta(232, 306) distribution
electors <- rbeta(10000, 232, 306)
# Create a data frame from the electors data
electors_df <- data.frame(electors)
# Create a density plot using ggplot2
ggplot(electors_df, aes(x = electors)) +
```

Figure 1

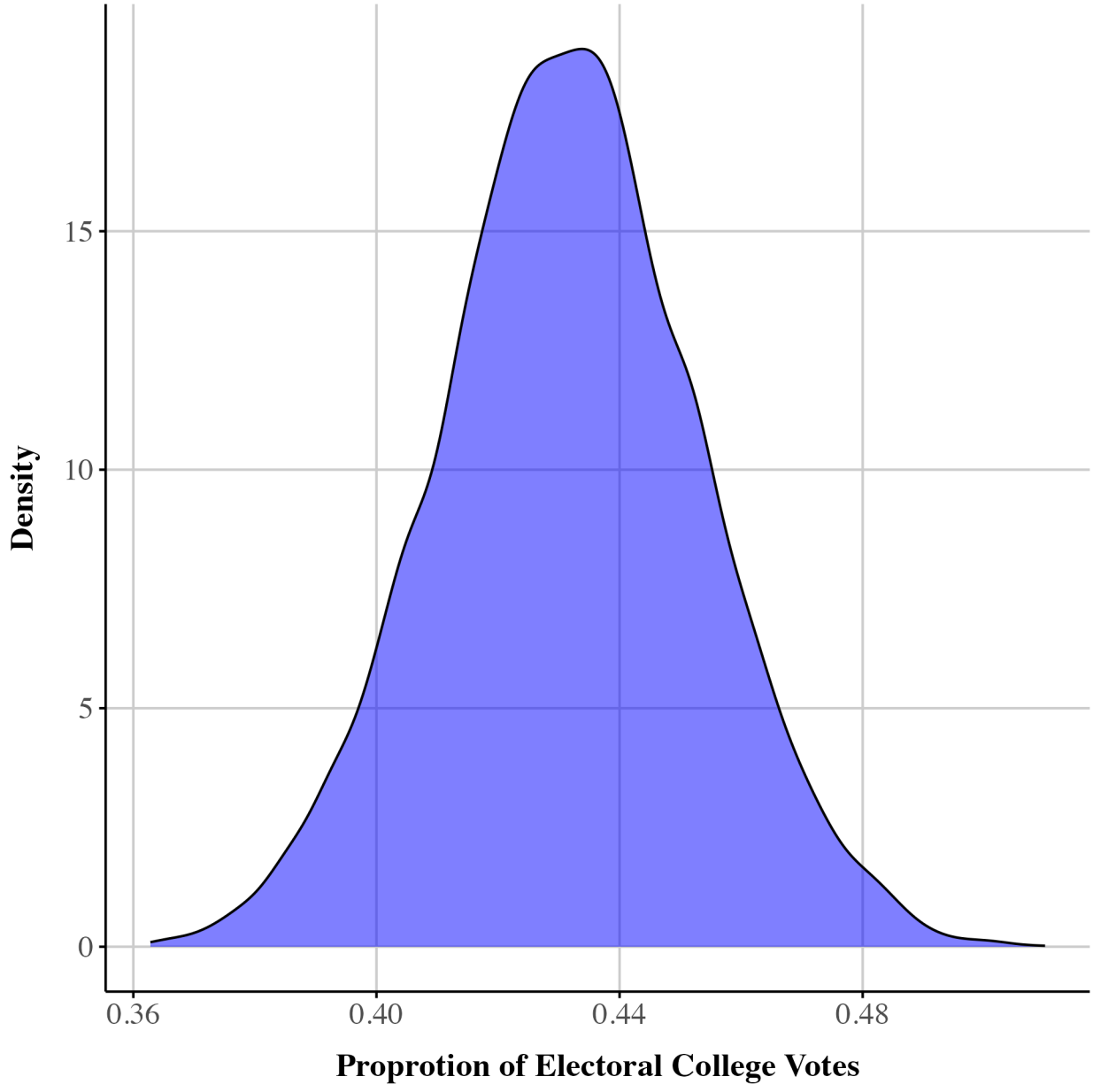
A. beta distribution with 4 correct and 2 incorrect responses on one test question. B. beta distribution with 2 correct and 4 incorrect responses on one test question



```
  geom_density(fill = "blue", alpha = 0.5) +
  labs(x = "Proprotion of Electoral College Votes", y = "Density") +
  theme_publication()
```

Figure 2

Density Plot of Electorial College Votes



What Figure 2 above shows is the range around the proportion of electors that Trump

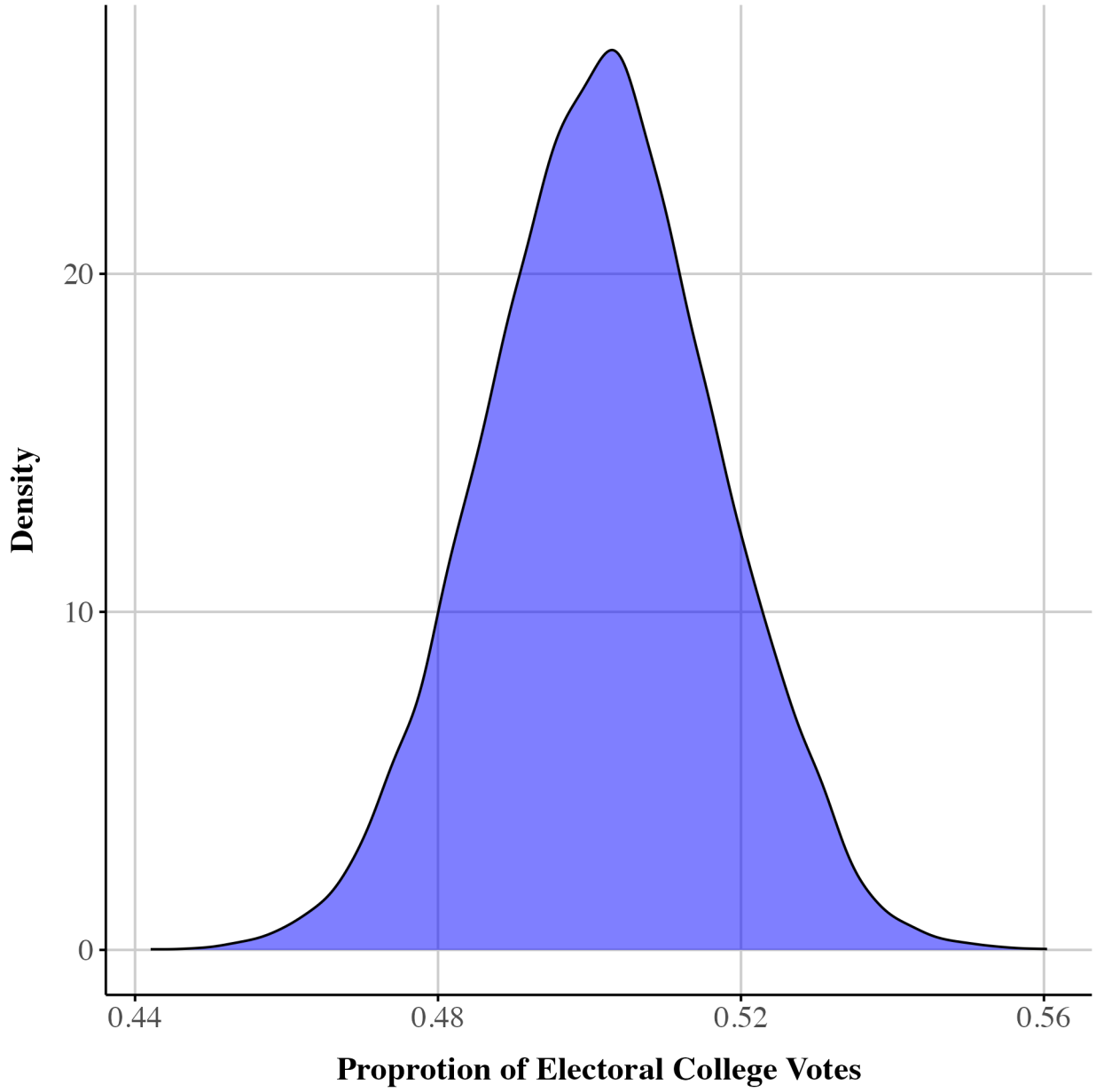
could win in 2024 if we use his 2020 performance as our prior knowledge. We would expect, based on chance alone, for his 2024 total to vary between 40% of electors to 48% of electors. In other words, he would be likely to still lose, but he could get a lot closer to a majority of electoral college votes. If we increase the sample size, say by including how many electoral votes Trump won in 2016 (304 out of 531), our uncertainty would decrease:

```
# Generate the electors data
electors <- rbeta(10000, 304 + 232, 227 + 306)
# Create a data frame from the electors data
electors_df <- data.frame(electors)


# Create a density plot using ggplot2
ggplot(electors_df, aes(x = electors)) +
  geom_density(fill = "blue", alpha = 0.5) +
   labs(x = "Proprotion of Electoral College Votes", y = "Density") +
  theme_publication()
```

Figure 3

Density Plot of Electors



The plausible range of percentages/probabilities of Figure 3 is now within 48% to 52% as taking into account Trump's 2016 results increases our sample size while also increasing our most likely estimate of his 2024 performance. Importantly, the beta distribution respects bounds.

Suppose, for example, that we only expect Trump to win 1 out of 538 electors. That would concentrate a substantial amount of plausible values at the lower end of the range of the variable:

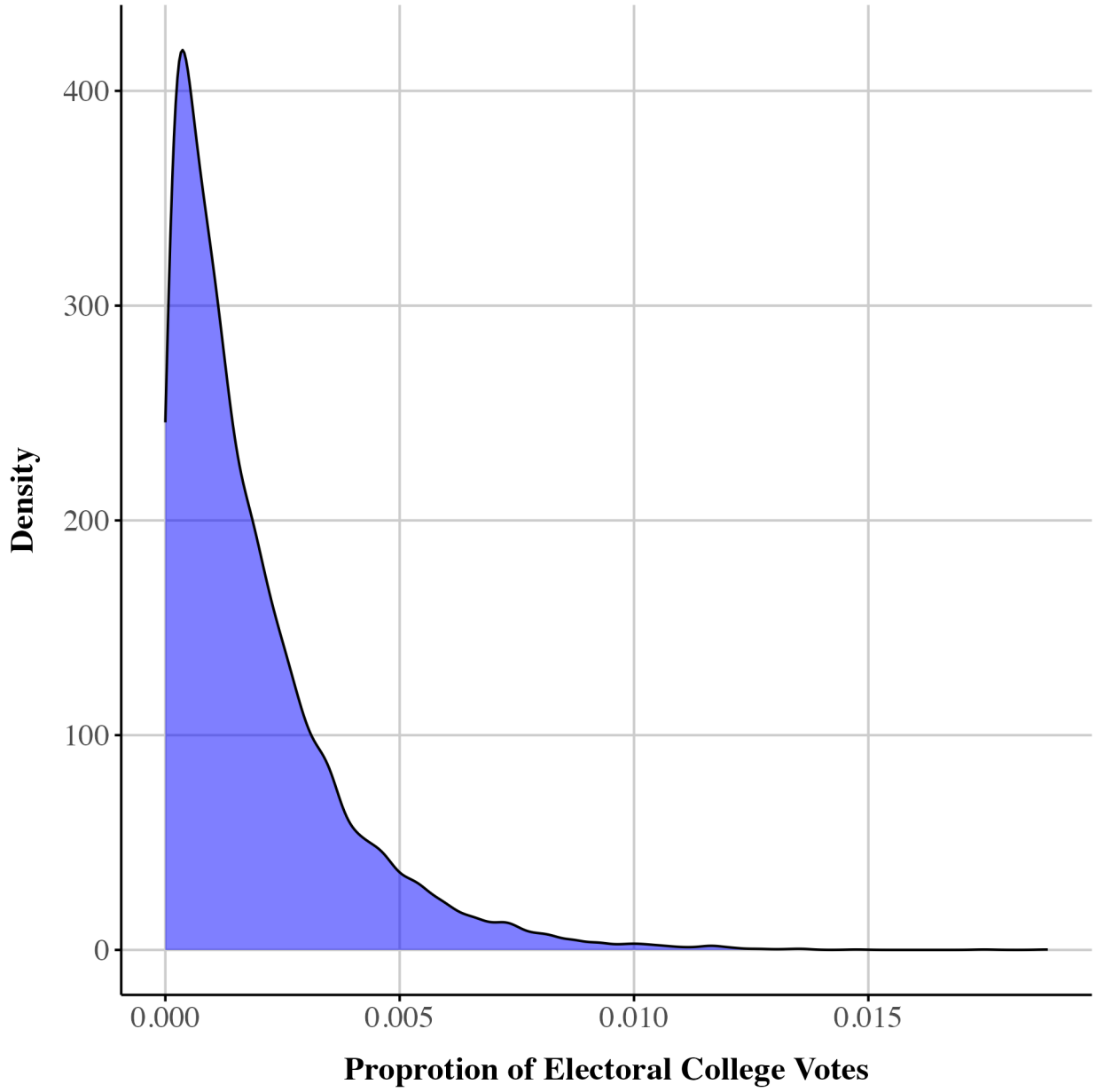
```r
electors <- rbeta(10000, 1, 537)


# Create a data frame from the electors data
electors_df <- data.frame(electors)


# Create a density plot using ggplot2
ggplot(electors_df, aes(x = electors)) +
  geom_density(fill = "blue", alpha = 0.5) +
   labs(x = "Proprotion of Electoral College Votes", y = "Density") +
  theme_publication()
```

Figure 4

Density Plot of Electors



What we can see in Figure 4 is a unique feature of the beta distribution: it respects upper and lower bounds. Here the beta distribution says it is most likely Trump wins a very small number of electoral college votes– below 1% of them – but it is plausible that he still wins more than 1%. It is impossible, as it should be, for Trump to win less than 0% of electors.

Mean and Precision

When talking about beta regression, instead of shape1 and shape2 or α or β we talk about μ and ϕ, where μ represents the mean or average, and ϕ represents the precision, in a roughly analogous way to how the Normal distribution has a mean and variance. We can reparameterize α and β into μ and ϕ via the following algebraic relationship

$$\text{Shape 1:} \quad a = \mu\phi \qquad\qquad \text{Mean:} \quad \mu = \frac{a}{a+b}$$

$$\text{Shape 2:} \quad b = (1-\mu)\phi \qquad\qquad \text{Precision:} \quad \phi = a + b$$

The variance can then be calculated as a function of μ and ϕ:

$$\frac{\mu \cdot (1-\mu)}{1+\phi}$$

Importantly, *the variance depends on the average value of the response*, which is what allows the model to non-linearly adjust to the bounds of the outcome.

Beta regression

We can use this parameterization of the beta distribution as a regression model of μ (mean) and ϕ (dispersion) parameters for a random variable that is continuous and bounded. While the beta distribution is presented for a variable that is between 0 and 1, it is straightforward to re-scale any random variable to 0 and 1 using the formula for normalization. Beta regression utilizes a logit link function to model the mean of the response variable as a function of the predictor variables (μ). Another link function, commonly the log (exponential) link, is used to model the dispersion parameter (ϕ). The application of these links ensures the parameters stay within their respective bounds, with μ between 0 and 1 and ϕ strictly positive. Overall, the beta regression approach respects the bounded nature of the data and allows for heteroskedasticity, making it highly appropriate for data that represents proportions or rates.

Motivating Example

Data and methods

Now that we have built up an intuition about the Beta distribution we can start to analyze some data. The principles of beta regression are best understood in the context of a real data set. The example we are gonna use comes from the learning and memory literature. A whole host of literature has shown extrinsic cues like fluency (i.e., how easy something is to process) can influence metamemory (i.e., how well we think we will remember something). As an interesting example, a line of research has focused on instructor fluency and how that influences both metamemory and actual learning. When an instructor uses lots of non-verbal gestures, has variable voice dynamics/intonation, is mobile about the space, and includes appropriate pauses when delivering content, participants perceive them as more fluent, but it does not influence actual memory performance, or what we learn from them (Carpenter et al., 2013; Toftness et al., 2017; Witherby & Carpenter, 2022). While fluency of instructor has not been found to impact actual memory across several studies, Wilford et al. (2020) found that it can. In several experiments, Wilford et al. (2020) showed that when participants watched multiple videos of a fluent vs. a disfluent instructor (here two videos as opposed to one), they remembered more information on a final test. Given the interesting, and contradictory results, we chose this paper to highlight. In the current tutorial we are going to re-analyze the final recall data from Wilford et al. (2021; Experiment 1a). In the spirit of open science, the authors made their data available here:https://osf.io/6tyn4/.

Accuracy data is widely used in psychology and is well suited for Beta regression. Despite this, it is common to treat accuracy data as continuous and unbounded, and analyze the resulting proportions using methods that fall under the general linear model. Below we will reproduce the analysis conducted by Wilford et al. (2020) (Experiment 1a) and then re-analyze it using Beta regression. We hope to show how Beta regression and its extensions can be a more powerful tool in making inferences about your data.

In Wilford et al. (2020) (Expt 1a), they presented participants with two short videos

highlighting two different concepts: (1) genetics of calico cats and (2) an explanation as to why skin wrinkles. Participants viewed either disfluent or fluent versions of these videos.[1] For each video, metamemory was assessed using JOLs. JOLs require participants to rate an item on scale between 0-100 with 0 representing the item will not be remembered and a 100 representing they will definitely remember the item. In addition, other questions about the instructor were assessed and how much they learned. After a distractor task, a final free recall test was given were participants had to recall as much information about the video as they could in 3 minutes. Participants could score up to 10 points for each video. Here we will only being looking at the final recall data, but you could also analyze the JOL data with a Beta regression.

Reanalysis of Wilford et al. Experiment 1a

GLM approach

In Experiment 1a, Wilford et al. (2020) only used the first time point (one video) and compared fluent and disfluent conditions with a *t*-test. They found better performance for participants watching the fluency instructor than the disfluency instructor (see Figure 5). In our re-analysis, we will also run a *t*-test, but in a regression context. This allows for easier generalization to the Beta regression approach that follows. Specifically, we will examine accuracy on final test as our DV (because the score was on a 10 point scale we divided by 10 to get a proportion) and look at Fluency (Fluent vs. Disfluent). Fluency was dummy coded with the fluent level serving as the reference level (coded as 0).

```
# Read the data
fluency_data <- fluency_data <- read.csv("data/miko_data.csv") %>%
  # Rename the columns for better readability
  rename(
    "Participant" = "ResponseID", # Rename "ResponseID" to "Participant"
```

[1] See an example of the fluent video here: https://osf.io/hwzuk. See an example of the disfluent video here: https://osf.io/ra7be.

```
    "Fluency" = "Condition", # Rename "Condition" to "Fluency"
    "Time" = "name", # Rename "name" to "Time"
    "Accuracy" = "value" # Rename "value" to "Accuracy"
  ) %>%
  # Transform the data
  mutate(
    Accuracy = Accuracy / 10, # Convert Accuracy values to proportions
    Fluency = ifelse(Fluency == 1, "Fluent", "Disfluent"), # rename levels
    Fluency_dummy = ifelse(Fluency == "Fluent", 0, 1), # Recode
    # Fluency: 1 becomes 0, others become 1
    Fluency_dummy = as.factor(Fluency_dummy), # turn fluency cond to dummy code
time=as.factor(Time))
```

Table 1

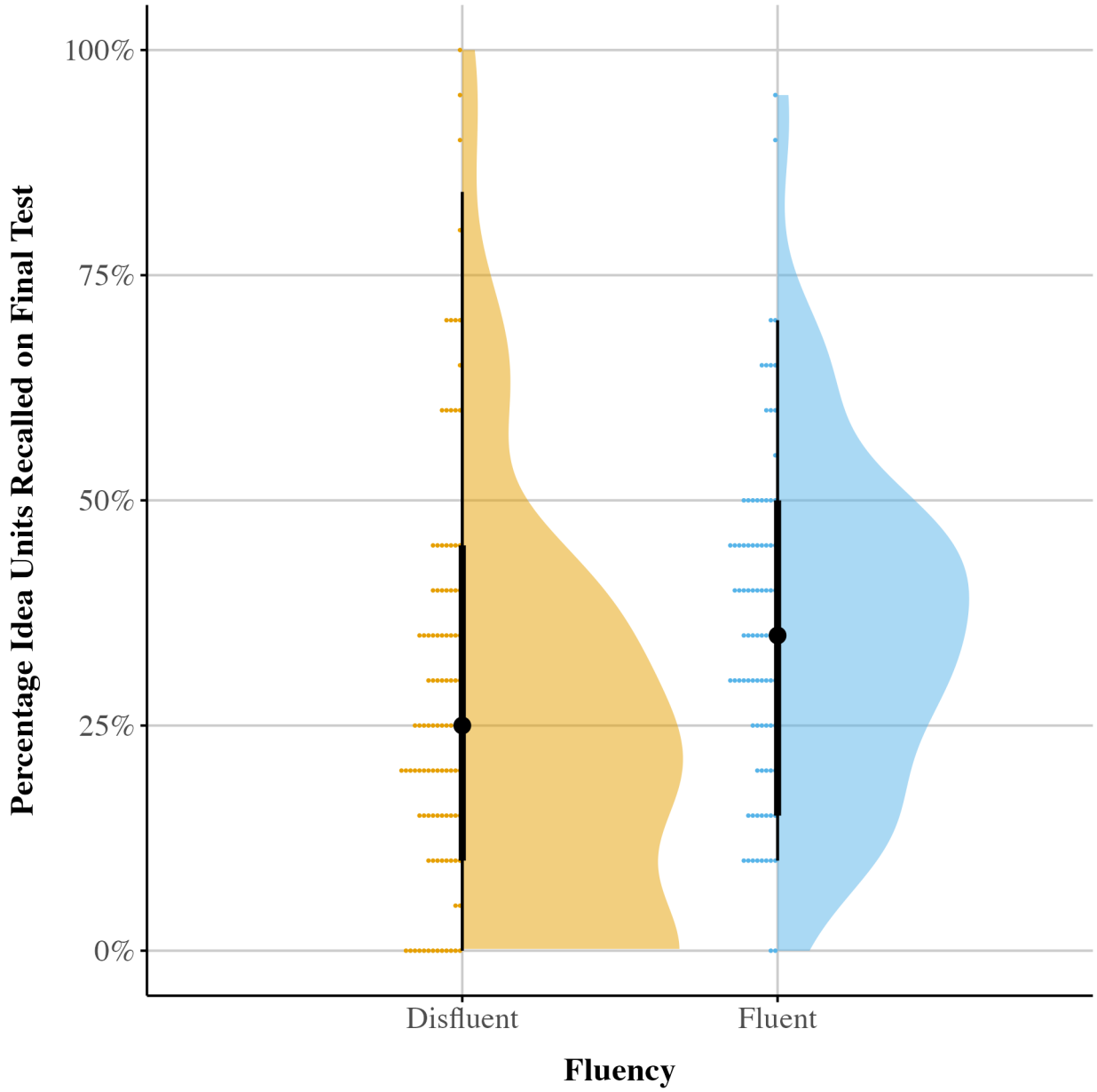
Dataset

X	Participant	Fluency	Time	Accuracy	Fluency_dummy	time
1	R_00Owxl28JtOADxn	Fluent	FreeCt2AVG	0.35	0	FreeCt2AVG
2	R_00Owxl28JtOADxn	Fluent	FreeCt1AVG	0.45	0	FreeCt1AVG
3	R_1DZHq7wW6PhBu7l	Fluent	FreeCt2AVG	0.30	0	FreeCt2AVG
4	R_1DZHq7wW6PhBu7l	Fluent	FreeCt1AVG	0.30	0	FreeCt1AVG
5	R_1FfS9t7o3G2waGp	Fluent	FreeCt2AVG	0.35	0	FreeCt2AVG
6	R_1FfS9t7o3G2waGp	Fluent	FreeCt1AVG	0.40	0	FreeCt1AVG

Load packages and data. As a first step, we will load the necessary packages along with the data we will be using. While we load all the necessary packages here, we also highlight when packages are needed as code chunks are run.

Figure 5

Raincloud plot for proprotion recalled on final test as a function of fluency



```
# packages needed
library(tidyverse) # tidy functions/data wrangling/viz
library(easystats)
library(scales) # percentage
library(tinytable) # tables
library(marginaleffects) # marginal effects
library(extraDistr) # Use extra distributions like dprop()
library(brms) # bayesian models
library(posterior) # cutpoints
library(bayesplot)
library(ggdist) # stat_halfeye viz bayes


options(scipen = 999) # get rid of scienitifc notation
```

Next, we read in the data from the repo, but you can also read it the directly from this url: https://raw.githubusercontent.com/jgeller112/beta_regression_tutorial/refs/heads/main/data/miko_data.csv?token=GHSAT0AAAAAAC4SIEVH762DOKQSI2F52YXYZ4AIT3Q. Before we begin, we have to make some modifications to the original dataset. Namely, we rename the columns to make them more informative, and transform the data. Here we transform accuracy so it is a proportion by multiplying each score by dividing by 100 . Finally, we dummy code the `Fluency` variable (`Fluency_dummy`) setting the fluent condition to 0 and the disfluent condition to 1. The first few rows of the data are displayed in Table 1 along with the data dictionary in Table 2.

```
# fit ols reg
ols_model <- lm(Accuracy ~ Fluency_dummy, data = fluency_data)
```

Table 3 displays the summary of the OLS regression model. The table shows two coefficients. The `Intercept` value refers to the accuracy in the fluent condition (because we dummy coded our variable). The `Fluency` coefficient (`Fluency_dummy1`) highlights the

Table 2

Data dictionary

Column	Key
Participant	Participant ID number
Fluency	Fluent vs. Disfluent
FluencyDummy	Fluent: 0; Disfluent: 1
Accuracy	Proportion recalled (idea units)

Table 3

OLS regression model coefficents

Parameter	Coefficient	SE	95% CI	p
(Intercept)	0.359	0.022	[0.316, 0.402]	<0.001
Fluency_dummy1	-0.072	0.03	[-0.13, -0.013]	0.017

difference between the fluent and disfluent condition, which is reliable, b = -0.072 , SE = 0.03 , 95% CIs = [-0.13,-0.013], p = 0.017. The results reproduce the findings from Wilford et al. (2020) —better memory when watching the fluent instructor video.

Bayesian regression

To fit our Bayesian models, we will be using a Bayesian package called `brms` (Bürkner, 2017) . `brms` is a powerful and flexible Bayesian regression modeling package that offers built in support for the beta distribution and some of the alternatives we discuss in this tutorial. This reduces the number of different packages researchers have to load in to their environment and makes it easier to build more complex models with similar syntax.

Adopting a Bayesian framework often provides more flexibility and allows us to quantity uncertainty around our estimates which makes it more powerful than the frequentist/MLE alternative. For the purposes of this tutorial, we will not be getting into the minutiae of Bayesian

data analysis (i.e., setting informative priors, MCMC sampling, etc,). For a more in-depth look into Bayesian data analysis I refer the reader to McElreath (2020) .

For the following analyses we will be using default priors provided by `brms`, which are non-informative or weak. This will get us something tantamount to a frequentist model with maximum likelihood estimates most of the readers should be familiar with.

We first start out by recreating the regression model from above in `brms` by using the `brm()` function and fitting a model looking at final test accuracy (`Accuracy`) as a function of instructor fluency (`fluency_dummy`). The syntax is similar to the `lm` function used above. Here we are concerned at modeling mean performance differences between the fluency and disfluency conditions.

We first start by loading the `brms` (Bürkner, 2017) and `cmdstanr` (Gabry et al., 2024) packages. We use the `cmdstanr` backend for Stan (Team, 2023) because it's faster and more modern than the defaults used to run models. In order to use the `cmdstanr` backend you will need to first install the package and also run `cmdstanr::install_cmdstan()` if you have not done so already.

```
# load brms and cmdstanr
library(brms)
library(cmdstanr)
```

```
cmdstanr::install_cmdstan()
```

Then we fit the model using the `brm()` function. The below model is run with four chains of 2000 Markov chain Monte Carlo iterations. For each chain, there was a 2000-iteration warm-up. The output of these models provides estimates of each effect (the coefficient `b_`, which is the mean of the posterior distribution), it's estimation error (the standard deviation of the posterior distribution), and its 95% credible interval (Crl). We inferred that there was evidence of an effect when the 95% Crl estimates did not include zero. Additional arguments were set to speed up the fitting of the models (`cores` = 4 and `backend` = `cmdstanr`) which we will not explain herein.

```
# fit ols reg
bayes_ols_model <- brm(Accuracy ~ Fluency_dummy, data = fluency_data,
 chains = 4,
  iter = 2000,
  warmup = 1000,
  cores = 4,
  seed = 1234,
  backend = "cmdstanr",
  file = "model_ols_bayes")
```

Table 4

Bayesian regression model coefficents

Parameter	Mean	95% CrI	pd
b_Intercept	0.359	[0.316, 0.403]	1
b_Fluency_dummy1	-0.072	[-0.131, -0.012]	0.99

Table 4 displays the summary of the Bayesian regression model. To make the output more readable, each model parameter is labeled with a prefix before the variable name. Comparing results with the OLS model we ran above our results are very similar. Additionally, the parameters can be interpreted in a similar manner. There are some notable differences, such as the absence of *t*- and *p*-values. There is a metric in the table that is included with models fit when using the `bayestestr` function from `easystats` called probability of direction (pd) that gives an indication of how much of the posterior distribution estimate is in one direction (positive or negative). The pd measure appears to correlated with *p*-values (see (Makowski, Ben-Shachar, & Lüdecke, 2019; Makowski, Ben-Shachar, Chen, et al., 2019)). A *pd* of **95%**, **97.5%**, **99.5%** and **99.95%** correspond approximately to two-sided *p*-value of respectively **.1**, **.05**, **.01** and **.001**. In addition to the *pd* value, one can look at the 95% credible interval (sometimes called highest

probablity density) to see if it includes 0–if it does not then the effect can be said to be significant. In the table below, the 95% credible intervals are located in the `CI_low` and `CI_high` columns.

The `b_Intercept` value refers to the accuracy in the fluent condition (because we dummy coded our variable). The Fluency coefficient (`b_Fluency_dummy1`) highlights the difference between the fluent and disfluent conditions, b = -0.072, 95% CrIs = [-0.131,-0.012], pd =1. These results map onto the results from the regression analysis above.

Beta regression

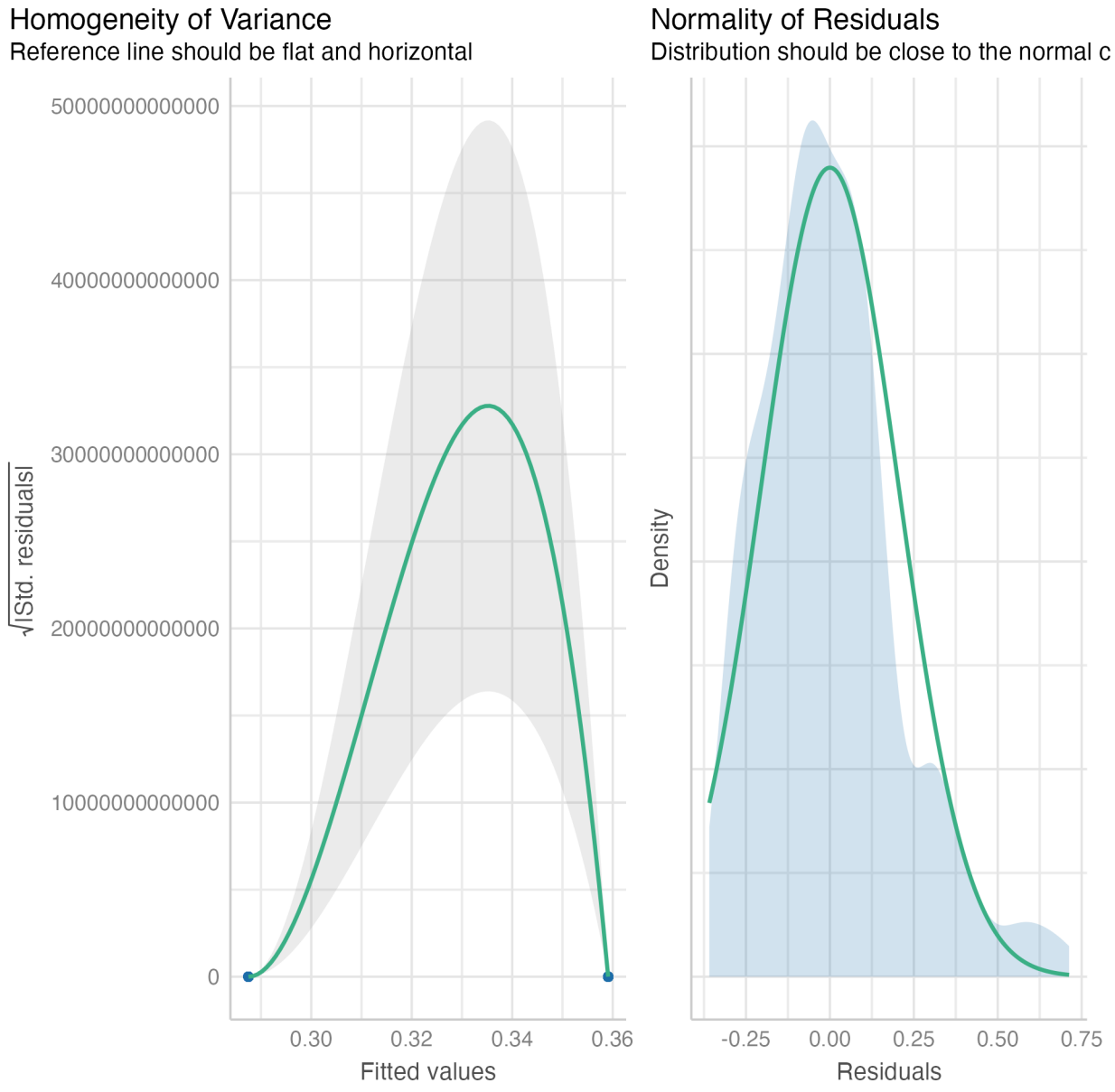
Wilford et al. (2020), using a traditional OLS approach, observed that instructor fluency impacts actual learning (p = 0.048). Keep in mind the traditional approach assumes normality of residuals and homoscadacity or constant variance. These assumptions are tricky to maintain when the continuous response approaches either the upper or lower boundary of the scale and are almost never true (see (Sladekova & Field, 2024). Does the model `ols_model_new` meet those assumptions? Using `easystats` (Lüdecke et al., 2022) and the `check_model()` function, we can easily check this. In Figure 6 , we see there definitely some issues with our model. Specifically, there appears to be violations of normality (right figure) and constant variance (homogeneity) (left-hand figure).

Given the outcome variable is proportional, one solution would be to run a beta regression model.

We can create the beta regression model in `brms`. In `brms`, we model each parameter independently. Recall from the introduction that in a Beta model we fit two parameters—μ and ϕ. We can easily do this by using the `bf()` function from `brms`. `bf()` facilitates the specification of several sub-models within the same formula call. We fit two formulas, one for μ and one for ϕ and store it in the `model_beta_bayes` object below. In the below `bf()` call, we are modeling Fluency as a function of Accuracy only for the μ parameter. For the ϕ parameter, we are only modeling the intercept value. This is saying dispersion does not change as a function of fluency.

Figure 6

Two assumption checks for our OLS model: Normality (left) and Homogeneity (right)



```
# fit model with mu and phi
model_beta_bayes <- bf(
  Accuracy ~ Fluency_dummy, # fit mu model
  phi ~ 1 # fit phi model
)
```

If we try to run `beta_brms` we get an error: `Error: Family 'beta' requires response greater than 0`. This is by design. If your remember, the Beta distribution can only model responses in the interval [0-1], but not responses that are exactly 0 or 1. We need make sure there are no zeros and ones in our dataset.

```
# run model


beta_brms <- brm(model_beta_bayes,
  data = fluency_data,
  family = Beta(),
  chains = 4,
  iter = 2000,
  warmup = 1000,
  cores = 4,
  seed = 1234,
  backend = "cmdstanr",
  file = "model_beta_bayes_reg" # Save this so it doesn't have to always rerun
)
```

Looking at our dataset (@tbl-01s) shows we have 9 rows with accuracy of 0, and 1 row with an accuracy of exactly 1. To run a beta regression, a common hack is to nudge our 0s towards .01 and our 1s to .99 so they fall within the interval of [0-1]. In the below code we use the

Table 5

Number of zeros and ones in our dataset

Accuracy	Count
0	15
1	1

`ifelse()` function from `tidyverse` package (Wickham, 2017) to change our zeros to .01 and ones to .99.

```
# transform 0 to 0.1 and 1 to .99
data_beta <- fluency_data %>%
  mutate(
    Accuracy = ifelse(Accuracy == 0, .01, ifelse(Accuracy == 1, .99, Accuracy))
  )
```

We can rerun our model replacing `fluency_data` with `data_beta` in the below function call.

```
beta_brms <- brm(model_beta_bayes,
  data = data_beta,
  family = Beta(),
  chains = 4,
  iter = 2000,
  warmup = 1000,
  cores = 4,
  seed = 1234,
  backend = "cmdstanr",
  file = "model_beta_bayes_reg_1" # Save this so it doesn't have to always rerun
```

```
)
```

No errors this time! We will perform the Beta regression using the nudged values of .01 and .99 values.

Squeezing lemons. As an aside, a more formal way to nudge these values is apply this formula proposed by Smithson and Verkuilen (2006):[2]

$$(y_i * (n - 1) + 0.5)/n$$

This involves adjusting the values of each value y and weighting it based on the total number of observations (N) and adding a small constant (0.5) to reduce bias or refine the estimate. While recent papers have used this approach (Alves et al., 2024), it is not advised. In particular, Kubinec (2022) shows that this formula can result in serious distortion of the outcome as the sample size N grows larger, resulting in ever smaller values that are "nudged". Because the beta distribution is a non-linear model of the outcome, values that are very close to the boundary, such as 0.00001 or 0.99999, will be highly influential outliers.

```
# code to use formula by smithson to nudge values
# taken from blog

# Define the transformation function for a dataframe
apply_transformation_to_df <- function(df, column_name) {
  n <- nrow(df) # Total number of rows (observations)

  df <- df %>%
    mutate(!!sym(column_name) := (!!sym(column_name) * (n - 1) + 0.5) / n)
```

[2] For those doing eye-tracking work, this called empirical logit.

```
  return(df)
}
```

```
transformed_df <- apply_transformation_to_df(fluency_data, "Accuracy")
```

Model parameters

Table 6 displays the summary from our Bayesian implementation. [3] The μ parameter estimates, which are labeled with an underscore `b_`while ϕ parameter coefficients are tagged with `b_phi` in the Parameter column.

Table 6

Posterior summary for beta regression using `brms`

Parameter	Mean	CI	CI_low	CI_high	pd	Rhat	ESS
b_Intercept	-0.524	0.95	-0.716	-0.323	1	0.999	3218
b_phi_Intercept	1.257	0.95	1.068	1.44	1	1	3314
b_Fluency_dummy1	-0.388	0.95	-0.66	-0.129	0.999	1	3650

Mean μ parameter

Let's look at our μ parameter first.

Table 7

μ parameter posterior summary

Parameter	Mean	95% CrI	pd
b_Intercept	-0.524	[-0.716, -0.323]	1
b_Fluency_dummy1	-0.388	[-0.66, -0.129]	0.999

[3] We have chain diagnostics included like Rhat and ESS which indicates how the MCMC sampling performed. For more information check out Gelman et al., 2013; Kruschke, 2014; McElreath, 2020)

In Table 7 the first set of coefficients (first two rows in the table) represent how factors influence the μ parameter, which is the mean of the beta distribution. These coefficients are interpreted on the scale of the logit, meaning they represent linear changes on a nonlinear space. The intercept term `(b_Intercept)` represents the log odds of the mean on accuracy for the fluent instructor condition.

Predicted probabilities. Parameter estimates are usually difficult to intercept on their own. We argue that readers should not spend too much time interpreting single model estimates. Instead they should discuss the effects of the predictor on the actual outcome of interest (in this case the 0-1 scale). The logit link allows us to transform back and forth between the scale of a linear model and the nonlinear scale of the outcome, which is bounded by 0 and 1. By using the inverse of the logit, we can easily transform our linear coefficients to obtain average effects on the scale of the proportions or percentages, which is usually what is interesting to applied researchers. In a simple case, we can do this manually, but when there are many factors in your model this can be quite complex.

To help us extract predictions from our model we will use a package called **`marginaleffects`** . [4] To get the proportions for each of our categorical predictors on the mu parameter we can use the function from the package called `predictions()`. These are displayed in Table 8.

```
# load marginaleffects package
library(marginaleffects)
```

```
predictions(beta_brms,
  newdata = datagrid(Fluency_dummy = c("0", "1"))
)
```

For the `Fluency` factor, we can interpret the estimate column in terms of proportions or

[4] `ggeffects` is another great package to extract marginal effects and plot (Lüdecke, 2018)

Table 8

Predicted probablities for fluency factor

Fluency_dummy	Mean	95% CrI
Fluent	0.37	[0.328, 0.42]
Disfluent	0.29	[0.248, 0.326]

percentages. That is, participants who watched the fluent instructor scored on average 36% on the final exam compared to 26% for those who watched the disfluent instructor.

We can also easily visualize these from `marginaleffects` using the `plot_predictions` function. After using this function, the proportions are visualized in Figure 7.

```
beta_plot <- plot_predictions(beta_brms, condition = "Fluency_dummy")
```

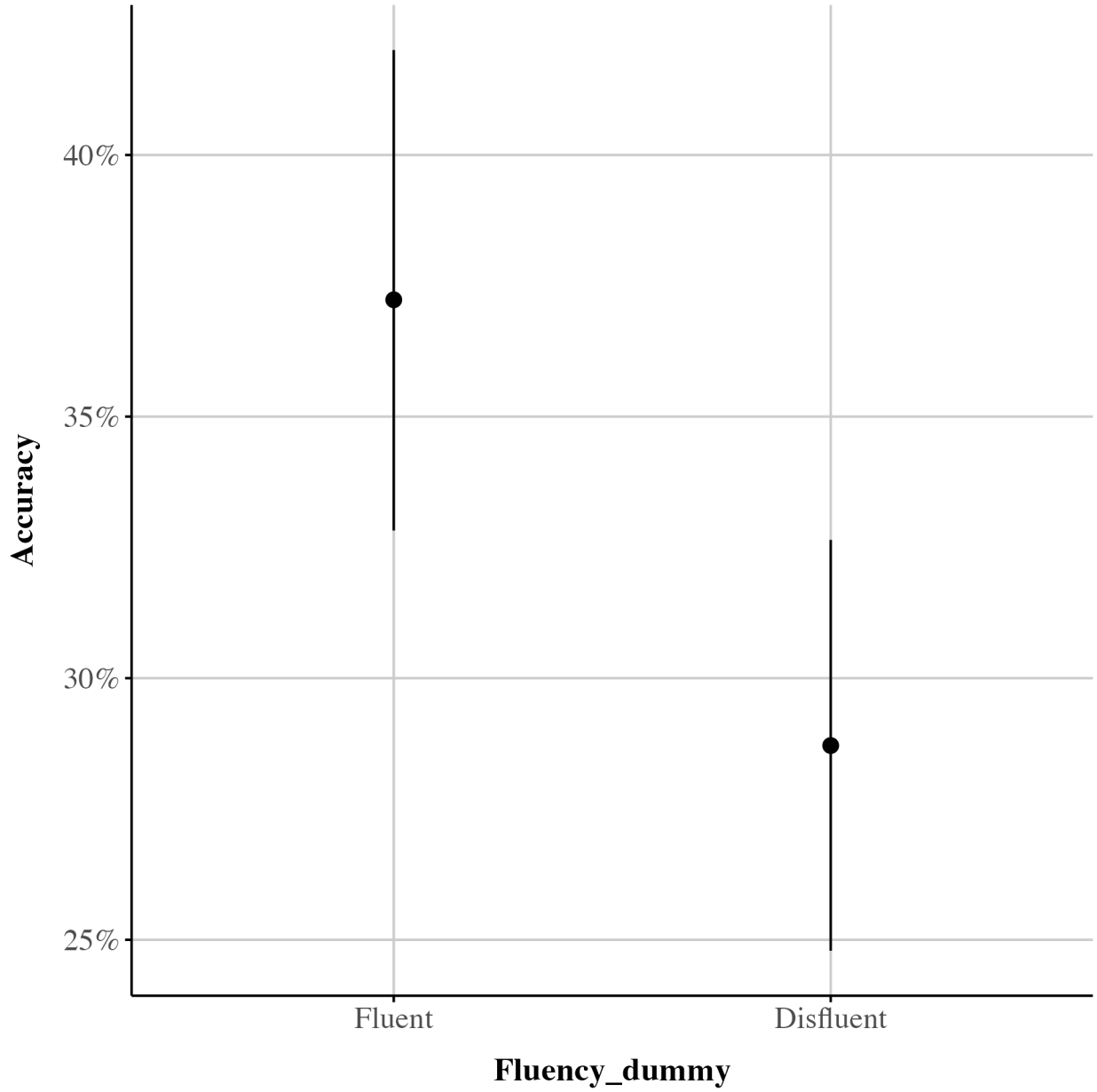
Marginal effects. Marginal effects provide a way to understand how changes in a predictor influence an outcome, holding all other factors constant in a specific manner. Technically, marginal effects are calculated using partial derivatives for continuous variables or finite differences for categorical and continuous variables, depending on the nature of the data and the research question. Substantively, these effects translate regression coefficients into a form that can be interpreted directly on the outcome scale of interest.

There are various types of marginal effects, and their calculation can vary across software packages. For example, the popular `emmeans` package computes marginal effects by holding all predictors at their means. In this tutorial, we will use the `marginaleffects` package, which focuses on average marginal effects (AMEs) by default. AMEs summarize effects by generating predictions for each row of the original dataset and then averaging these predictions. This approach retains a strong connection to the original data while offering a straightforward summary of the effect of interest.

One practical application of AMEs is calculating the risk difference for categorical variables. The risk difference represents the difference in average outcomes between two groups

Figure 7

Predicted probablities for fluency factor



or conditions. Using the `avg_comparisons()` function in the `marginaleffects` package, we can compute this metric directly. By default, the function calculates the discrete difference between groups. It can also compute other effect size metrics, such as odds ratios and risk ratios, depending on the research question (see @tbl-or for more details). This flexibility makes it a powerful tool for interpreting regression results in a meaningful way.

```
# get risk difference by default
beta_avg_comp <- avg_comparisons(beta_brms)
```

Table 9

Risk difference for fluency

Term	Contrast	Mean	95% CrI
Fluency_dummy	1 - 0	-0.085	[-0.146, -0.028]

Table 9 displays the risk difference for the fluency factor (`estimate` column). The difference between the fluent and disfluent conditions is .10. That is, participants who watched a fluent instructor scored 10% higher on the final recall test than participants who watched the disfluent instructor. However, Our credible interval [-.0174, -0.011] does not include zero so we can say it is statistically significant.

We can also get the odds ratio with `avg_comparisons` (see Table 10).

```
# get odds ratios as an example
avg_comparisons(beta_brms,
  comparison = "lnoravg",
  transform = "exp"
) %>%
  select(-predicted_lo, -predicted_hi, -tmp_idx, -predicted)%>%
  mutate(
```

```
   mutate(mutate(across(where(is.numeric), ~ round(.x, 3)))),
         `95% CrI` = str_glue("[{conf.low}, {conf.high}]")) %>%
  rename("Contrast" = "contrast","Mean" = "estimate", "Term" = "term") %>%
  select(Term, Contrast, Mean,`95% CrI`) %>%
  tt()  %>%
   format_tt(escape = TRUE)
```

Table 10

Odds ratio for fluency factor

Term	Contrast	Mean	95% CrI
Fluency_dummy	ln(odds(1) / odds(0))	0.679	[0.517, 0.879]

> **Effect Size**
>
> I was think we could show how to calculate effect size measure (Cohen's h or d) for the marginal difference in prob between the two conditions

In psychology it is common to report effect size measures like Cohen's D. When working with proportions we can calculate something similar called Cohen's h. Taking our proportions, we can use the below equation to calculate Cohen's h along with the 95% CrI around it.

$$h = 2 \cdot \left(\arcsin\left(\sqrt{p_1}\right) - \arcsin\left(\sqrt{p_2}\right)\right)$$

```
# Proportions
p1 <- 0.373  # Fluent
p2 <- 0.287  # Disfluent


# Cohen's h formula
```

```
cohens_h <- 2 * (asin(sqrt(p1)) - asin(sqrt(p2)))
round(cohens_h, 3)
```

```
[1] 0.183
```

```
#CI lower and upper bounds
p1_lower <- 0.328
p1_upper <- 0.419
p2_lower <- 0.250
p2_upper <- 0.329

# Lower bound of h
h_lower <- 2 * (asin(sqrt(p1_lower)) - asin(sqrt(p2_upper)))
# Upper bound of h
h_upper <- 2 * (asin(sqrt(p1_upper)) - asin(sqrt(p2_lower)))

round(c(h_lower, h_upper), 3)
```

```
[1] -0.002  0.361
```

Using this metric we see the effect size is small (0.183), 95% CrI [-0.0021292, 0.3608817].

Precision (ϕ) component

The other component we need to pay attention to is the dispersion or precision parameter coefficients labeled as 'B_phi in Table 11. The dispersion (ϕ) parameter tells us how precise our estimate is. Specifically, ϕ in beta regression tells us about the variability of the response variable around its mean. Specifically, a higher dispersion parameter indicates a narrower distribution, reflecting less variability. Conversely, a lower dispersion parameter suggests a wider distribution, reflecting greater variability. The main difference between a dispersion parameter and the

variance is that the dispersion has a different interpretation depending on the value of the outcome, as we show below. The best way to understand dispersion is to examine visual changes in the distribution as the dispersion increases or decreases.

Understanding the dispersion parameter helps us gauge the precision of our predictions and the consistency of the response variable. In `beta_brms` we only modeled the dispersion of the intercept. When ϕ is not specified, the intercept is modeled by default (see Table 11).

```
model_parameters(beta_brms, "mean") %>%
  filter(startsWith(Parameter, "b_phi")) %>%
  mutate(
    mutate(mutate(across(where(is.numeric), ~ round(.x, 3)))),
  `95% CrI` = str_glue("[{CI_low}, {CI_high}]")) %>%
  select(Parameter, Mean, `95% CrI`, pd)%>%
  tt() %>%
  format_tt(digits = 3) %>%
   format_tt(escape = TRUE)
```

Table 11

Dispersion parameter beta regression model

Parameter	Mean	95% CrI	pd
b_phi_Intercept	1.26	[1.068, 1.44]	1

The intercept under the precision heading is not that interesting. It represents the overall dispersion in the model. We can model the dispersion of the `Fluency` factor—this allows dispersion to differ between the fluent and disfluent conditions. To do this we add `Fluency_dummy` to the phi model in `bf()`. We model the precision of the `Fluency` factor by using a ~ and adding factors of interest to the right of it.

```
model_beta_bayes_disp <- bf(
  Accuracy ~ Fluency_dummy, # fit mu model
  phi ~ Fluency_dummy # fit phi model with fluency
)
```

```
beta_brms_dis <- brm(model_beta_bayes_disp,
  data = data_beta,
  family = Beta(),
  chains = 4,
  iter = 2000,
  warmup = 1000,
  cores = 4,
  seed = 1234,
  backend = "cmdstanr",
  file = "model_beta_bayes_dis_run" # Save this so it doesn't have to always rerun
)
```

```
Running MCMC with 4 parallel chains...

Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
```

```
Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
```

```
Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
```

```
Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1 finished in 0.6 seconds.
Chain 2 finished in 0.6 seconds.
```

```
Chain 3 finished in 0.6 seconds.
Chain 4 finished in 0.6 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.6 seconds.
Total execution time: 0.7 seconds.
```

Table 11 displays the model summary for the precision parameter. It is important to note that the estimates are logged and not on the original scale (this is only the case when additional parameters are modeled). To interpret them on the original scale, we can exponentiate the log-transformed value—this transformation gets us back to our original scale (see Table 12). In below model call, we set `exponentiate` = TRUE.

```
beta_model_dis_exp <- beta_brms_dis %>%
  model_parameters(exponentiate=TRUE, centrality = "mean")
```

Table 12

beta regression model summary for fluency factor with ϕ parameter exponentiated

Parameter	Mean	95% CrI	pd
b_phi_Intercept	4.894	[3.739, 6.371]	1
b_phi_Fluency_dummy1	0.573	[0.397, 0.826]	0.999

The ϕ intercept represents the precision of the fluent condition. The ϕ coefficient for `Fluency_dummy1` represents the change in that precision for performance between the fluent vs. disfluent conditions. The Cr.intervals [.235, .684] do not include 0 so our results are statistically significant.

It is important to note that these estimates are not the same as the marginal effects we discussed earlier. Changes in dispersion will change the shape of the distribution but not necessarily the average value of the response. This makes dispersion most interesting for research

questions that focus on other features of the distribution besides the mean, such as the level of polarization in an outcome.

A critical assumption of the GLM is homoscedasticity, which means constant variance of the errors. Here we see one of the benefits of a beta regression model: we can include a dispersion parameter for Fluency. Properly accounting for dispersion is crucial because it impacts the precision of our mean estimates and, consequently, the significance of our coefficients. The inclusion of dispersion in the our model changed the statistical significance of the μ coefficient (see Figure 8). This suggests that failing to account for the dispersion of the variables might lead to biased estimates. This highlights the potential utility of an approach like beta regression over a traditional approach as beta regression can explicitly model dispersion and address issues of heteroscedasticity.

We won't always need to include dispersion parameters for each of our variables. We advise conducting very simple model comparisons like leave one out (loo) cross validation to examine if a dispersion parameter should be considered in our model.

Posterior predictive check

It is a always a good idea to check how well your data fits the model. The `pp_check()` function allows us to examine the fit between our data and the model. In Figure 9, multiple draws or repetitions from the posterior distribution are plotted (light blue lines) against the observed data. (dark blue). Ideally, the predictive draws (the light blue lines) should show reasonable resemblance with the observed data (dark blue line). We see it does a pretty good job capturing the data. Ideally, the predictive draws (the light blue lines) should show reasonable resemblance with the observed data (dark blue line). We see it does a pretty good job capturing the data.

```
pp_check(beta_brms_dis, ndraws = 100)  + # ndraws is nu
  theme_publication()
```

Figure 8

Comparison of posterior distributions for the risk difference in fluency: Simple model (no dispersion for Fluency) vs. complex model with dispersion

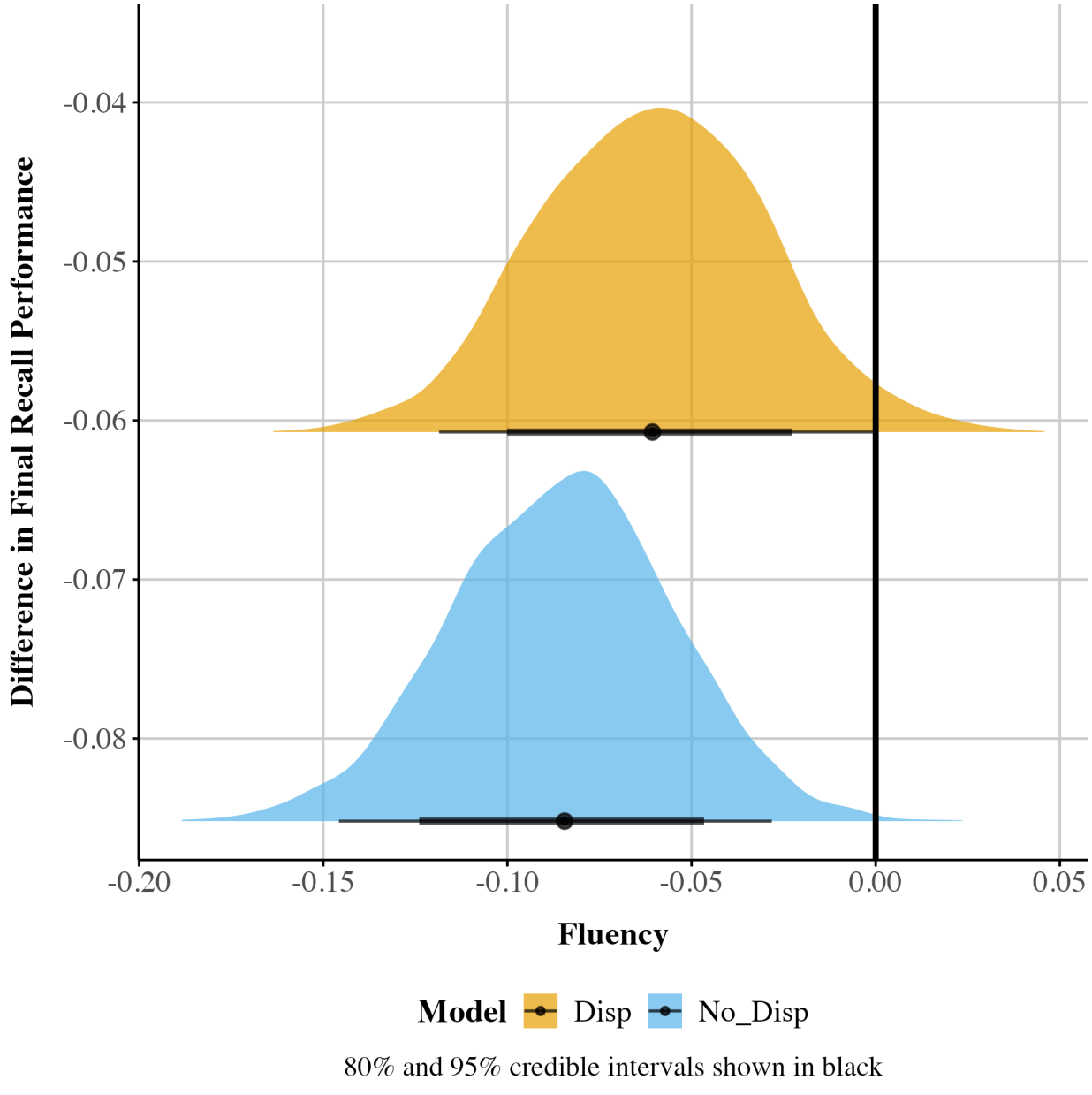
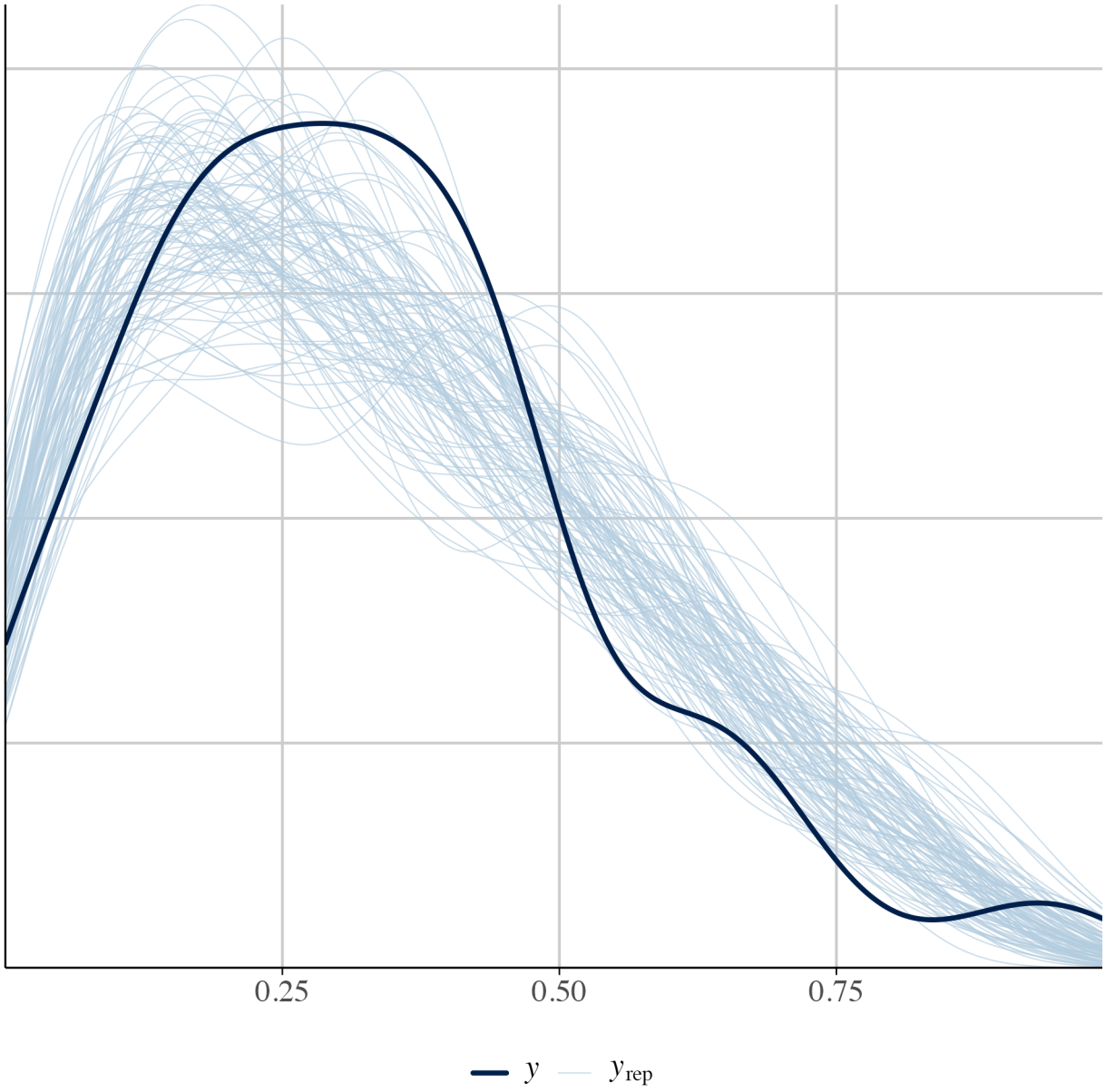


Figure 9

Posterior predictive check for our Beta model with 100 draws.



Zero-inflated beta (ZIB) regression

A limitation of the beta regression model is it can can only model values between 0 and 1, but not exactly 0 or 1. In our dataset we have 9 rows with `Accuracy` equal to zero.

To use the Beta distribution we nudged our zeros to 0.01–which is never a good idea in

practice. In our case it might be important to model the structural zeros in our data, as fluency of instructor might be an important factor in predicting the zeros in our model. There is a model called the zero-inflated beta (ZIB) model that takes into account the structural 0s in our data. We'll still model the μ and ϕ (or mean and precision) of the beta distribution, but now we'll also add one new special parameter: α.

With zero-inflated regression, we're actually modelling a mixture of the data-generating process. The α parameter uses a logistic regression to model whether the data is 0 or not. Substantively, this could be a useful model when we think that 0s come from a process that is relatively distinct from the data that is greater than 0. For example, if we had a dataset of with proportion of looks or eye fixatons to certain areas on marketing materials, we might want a separate model for those that do not look at certain areas on the screen because those folks who do not look might be substantively different than those that look. For example, in a recent study we

We can fit a ZIB model using `brms` and use the `marginaleffects` package to make inferences about our parameters of interest. Similar to our beta model we fit in `brms`, we will use the `bf()` function to fit several models. We fit our μ and ϕ parameters as well as our zero-inflated parameter (α; here labeled as `zi`). In `brms` we can use the zero_inflated_beta family.

```
# keep 0 but transform 1 to .99
data_beta_0 <- fluency_data %>%
  mutate(Accuracy = ifelse(Accuracy == 1, .99, Accuracy))
```

```
#|
# fit zero-inflated beta in brms
zib_model <- bf(
  Accuracy ~ Fluency_dummy, # The mean of the 0-1 values, or mu
  phi ~ Fluency_dummy, # The precision of the 0-1 values, or phi
  zi ~ Fluency_dummy, # The zero-or-one-inflated part, or alpha
  family = zero_inflated_beta()
```

```
)
```

Below we pass `zib_model` to the `brm` function.

```
fit_zi <- brm(
  formula = zib_model,
  data = data_beta_0,
  cores = 4,
  iter = 2000,
  warmup = 1000,
  seed = 1234,
  backend = "cmdstanr",
  file = "model_beta_bayes_zib"
)
```

Table 13

Model summary (posterior distribution) for zero-inflated beta model

Parameter	Mean	95% CrI	pd
b_Intercept	-0.515	[-0.675, -0.348]	1
b_phi_Intercept	1.771	[1.471, 2.053]	1
b_Fluency_dummy1	-0.097	[-0.347, 0.154]	0.776
b_phi_Fluency_dummy1	-0.348	[-0.731, 0.04]	0.961
b_zi_Intercept	-3.783	[-5.461, -2.563]	1
b_zi_Fluency_dummy1	1.853	[0.48, 3.635]	0.999

Predicted probabilities and marginal effects

Table 13 provides a summary of the posterior distribution for each parameter. As stated before, it is preferable to back-transform our estimates to get probabilities. We can model the parameters separately using the `dpar` argument setting to: μ, ϕ, α. To get the predicted probabilities we can use the `avg_predictions()` function and to get risk difference between the levels of each factor we can use the `avg_comparisons()` function from `marginaleffects` package. Here we look at the risk difference for Fluency under each parameter.

Mean (μ). Looking at Table 14 there is no significant effect for Fluency.

```
marg_zi_brms <- fit_zi %>%
  avg_comparisons(
    variables = "Fluency_dummy",
    dpar = "mu",
    comparison = "difference"
  )
```

Table 14

Risk difference (mu) for fluency factor

Term	Contrast	Mean	95% CrI
Fluency_dummy	1 - 0	-0.022	[-0.08, 0.036]

Dispersion (ϕ). Looking at Table 15, there is a significant effect of fluency on dispersion, with disfluency having a larger variation than fluency.

```
marg_phi_brms <- fit_zi %>%
  avg_comparisons(
    variables = "Fluency_dummy",
    dpar = "phi",
```

```
    comparison = "difference"
  )
```

Table 15

Risk difference (Phi) for fluency factor

Term	Contrast	Mean	95% CrI
Fluency_dummy	1 - 0	-1.75	[-3.886, 0.189]

Zero-inflated *(α)*

In Table 16 we see that watching a lecture video with a fluent instructor reduces the proportion of zeros by about 13%. The CrI does not include zero.

```
marg_zi_brms <- fit_zi %>%
  avg_comparisons(
    variables = "Fluency_dummy",
    dpar = "zi",
    comparison = "difference"
  )
```

Table 16

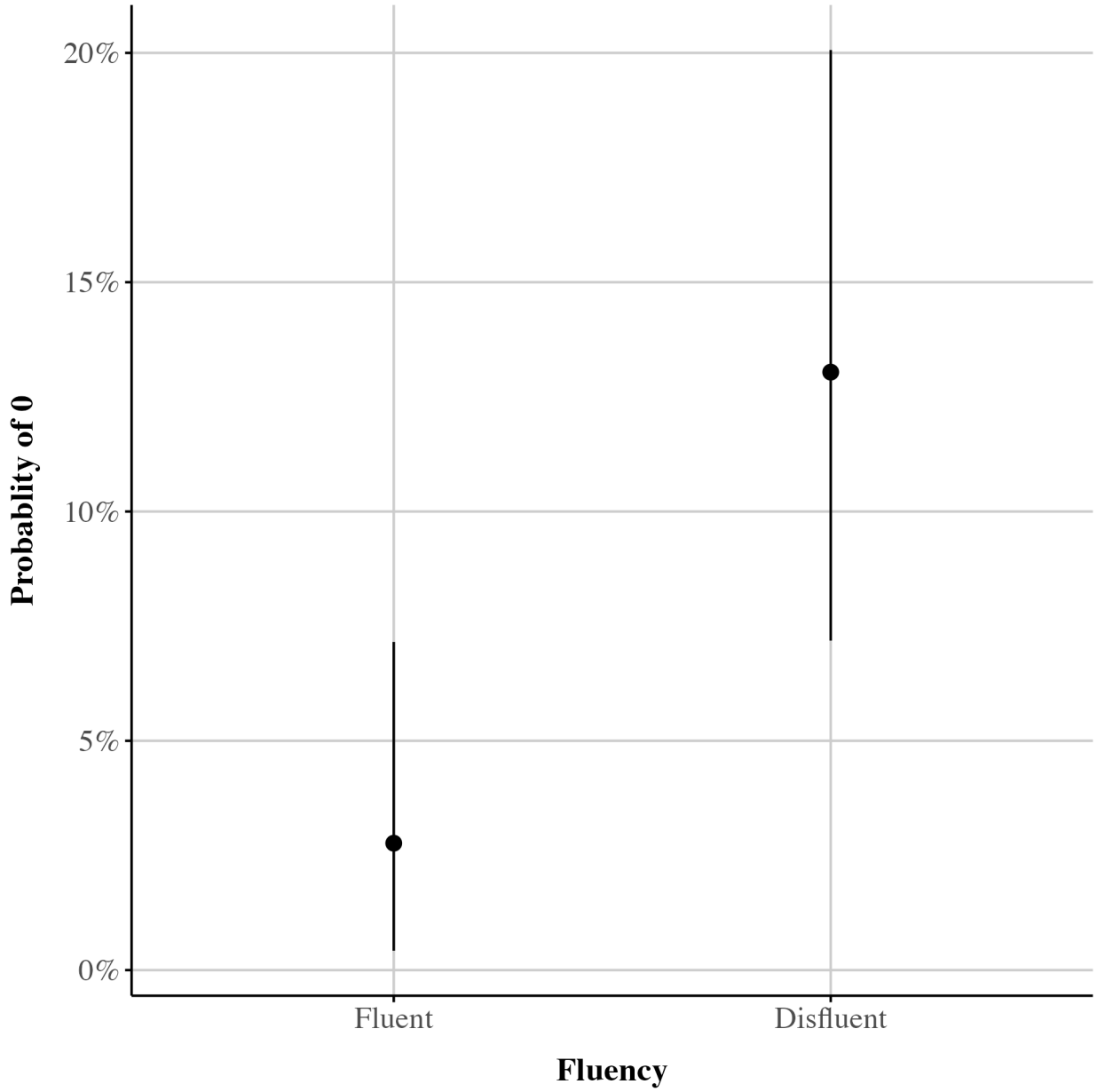
Posterior summary of the risk difference (zero-inflated) for fluency factor

Term	Contrast	Mean	95% CrI
Fluency_dummy	1 - 0	0.103	[0.033, 0.178]

If we wanted we could easily visualize the zero-inflated part of the model (see Figure 10).

Figure 10

Visualization of predicted probablites for zero-inflated part of model



Zero-one-inflated beta (ZOIB)

The ZIB model works well if you have 0s in your data, but not 1s.[5] Sometimes it is theoretically useful to model both zeros and ones as separate processes or to consider these values as essentially similar parts of the continuous response, as we show later in the ordered beta regression model. For example, this is important in visual analog scale data where there might be a prevalence of responses at the bounds (Kong & Edwards, 2016), in JOL tasks (Wilford et al., 2020), or in a free-list task where individuals provide open responses to some question or topic which are then recoded to fall between 0-1 (Bendixen & Purzycki, 2023). Here 0 and 1 are meaningful; 0 means item was not listed and 1 means item was listed first.

In our data, we have exactly one value equal to 1. While probably not significant to alter our findings, we can model ones with a special type of model called the zero-one-inflated beta (ZOIB) model if we believe that both 0s and 1s are distinct outcomes. Returning to our looking example from earlier, it is possible for a proportion of looks to be one, especially if participants are only looking at one area of interest and nothing else.

Similar to our beta and zero-inflated models above, we can fit a ZOIB model in `brms` quite easily using the `zero_one_inflated_beta family`. In this model, we fit four parameters or sub-models. We fit separate models for the mean (μ) and the precision (ϕ) of the beta distribution; a zero-one inflation parameter (i.e. the probability that an observation is either 0 or 1; α) from the zero-inflated model; and a 'conditional one inflation' parameter (i.e. the probability that, given an observation is 0 or 1, the observation is 1; γ). This specification captures the entire range of possible values while still being constrained between zero and one.

We use the `bf()` function again to fit models for our four parameters. We model each parameter as a function of Fluency.

---

[5] In cases where there are large amounts of 1s but no zeros you can fit a one-inflated model. Currently you can do so with the `gamlss` package in R

```
# fit the zoib model


zoib_model <- bf(
  Accuracy ~ Fluency_dummy, # The mean of the 0-1 values, or mu
  phi ~ Fluency_dummy, # The precision of the 0-1 values, or phi
  zoi ~ Fluency_dummy, # The zero-or-one-inflated part, or alpha
  coi ~ Fluency_dummy, # The one-inflated part, conditional on the 0s, or gamma
  family = zero_one_inflated_beta()
)
```

We then pass the `zoib_model` to our `brm()` function. The summary of the output is in Table 17 .

```
# run the zoib mode using brm


fit_zoib <- brm(
  formula = zoib_model,
  data = fluency_data,
  chains = 4, iter = 2000, warmup = 1000,
  cores = 4, seed = 1234,
  backend = "cmdstanr",
  file = "model_beta_zoib_1"
)
```

Model parameters

The output for the model is pretty lengthy (see Table 17)—we are estimating four parameters each with their own independent responses and sub-models. All the coefficients are on the logit scale, except ϕ , which is on the log scale. Thankfully drawing inferences for all these

Table 17

Model summary (posterior distribution) for the zero-one inflated beta model

Parameter	Mean	95% CrI	pd
b_Intercept	-0.515	[-0.677, -0.352]	1
b_phi_Intercept	1.763	[1.46, 2.045]	1
b_zoi_Intercept	-3.772	[-5.411, -2.62]	1
b_coi_Intercept	-2.766	[-9.153, 0.998]	0.906
b_Fluency_dummy1	-0.178	[-0.413, 0.063]	0.927
b_phi_Fluency_dummy1	-0.122	[-0.512, 0.268]	0.73
b_zoi_Fluency_dummy1	1.921	[0.605, 3.665]	0.999
b_coi_Fluency_dummy1	0.254	[-4.382, 6.901]	0.522

different parameters, plotting their distributions, and estimating their average marginal effects looks exactly the same—all the **brms** and **marginaleffects** functions we used work the same.

Predictions and marginal effects

With `marginaleffects` we can choose marginalize over all the sub-models, averaged across the 0s, continuous responses, and 1s in the data, or we can model the parameters separately using the `dpar` argument like we did above setting it to: μ, ϕ, α, γ. Using `avg_predictions()` and not setting `dpar` we can get the predicted probabilities across all the sub-models. We can also plot the overall difference between fluency and disfluency for the whole model with `plot_predictions`.

Ordered Beta regression

Looking at the output from the ZOIB model Table 17, we can see how running a model like this can become vastly complex and computational intensive as it is fitting sub-models for each parameter. The ability to consider 0s and 1s as distinct processes from continuous values

comes at a price in terms of complexity. A special version of the ZOIB was recently developed called ordered beta regression (Kubinec, 2022). The ordered beta regression model allows for the analysis of continuous data (between 0-1) and discrete outcomes (e.g., 0 or 1) without requiring that either be fully distinct from the other. In the simplest sense, the ordered beta regression model is a hybrid model that estimates a weighted combination of a beta regression model for continuous responses and a logit model for the discrete values of the response.

The weights that average together the two parts of the outcome (i.e., discrete and continuous) are determined by cutpoints that are estimated in conjunction with the data in a similar manner to what is known as an ordered logit model. An in-depth explanation of ordinal regression is beyond the scope of this tutorial (Bürkner & Vuorre, 2019; but see Fullerton & Anderson, 2021). At a basic level, ordinal regression models are useful for outcome variables that are categorical in nature and have some inherent ordering (e.g., Likert scale items). To preserve this ordering, ordinal models rely on the cumulative probability distribution. Within an ordinal regression model it is assumed that there is a continuous but unobserved latent variable that determines which of k ordinal responses will be selected. For example on a typical Likert scale from 'Strongly Disagree' to 'Strongly Agree', you could assume that there is a continuous, unobserved variable called 'Agreement'. While we cannot measure Agreement directly, the ordinal response gives us some indication about where participants are on the continuous Agreement scale. $k - 1$ cutoffs are then estimated to indicate the point on the continuous Agreement scale at which your Agreement level is high enough to push you into the next ordinal category (say Agree to Strongly Agree). Coefficients in the model estimate how much different predictors change the estimated *continuous* scale (here, Agreement). Since there's only one underlying process, there's only one set of coefficients to work with (proportional odds assumption). In an ordered beta regression, three ordered categories are modeled: (1) exactly zero, (2) somewhere between zero and one, and (3) exactly one. In an ordered beta regression, (1) and (2) are modeled with cumulative logits, where one cutpoint is the the boundary between Exactly 0 and Between 0 and 1 and the other cutpoint is the boundary between *Between 0 and 1*

and *Exactly 1*. Somewhere between 0-1 (3) is modeled as a beta regression with parameters reflecting the mean response on the logit scale. Ultimately, employing cutpoints allows for a smooth transition between the bounds and the continuous values, permitting both to be considered together rather than modeled separately as the ZOIB requires.

The ordered beta regression model has shown to be more efficient and less biased than some of the methods discussed (Kubinec, 2022) herein and has seen increasing usage across the biomedical and social sciences (Martin et al., 2024; Nouvian et al., 2023; Shrestha et al., 2024; Smith et al., 2024; Wilkes et al., 2024) because it produces only a single set of coefficient estimates in a similar manner to a standard beta regression or OLS.

Fitting an ordered Beta regression

To fit an ordered Beta regression in a Bayesian context we use the `ordbetareg` (Kubinec, 2023) package. `ordbetareg` is a front-end to the `brms` package that we described earlier; in addition to the functions available in the package, most `brms` functions and plots, including the diverse array of regression modeling options, will work with `ordbetareg` models.

We first load the `ordbetareg` package. You can download it from CRAN or from here: https://github.com/saudiwin/ordbetareg_pack.

```
# load ordbetareg package
library(ordbetareg)
```

The `ordbetareg` package uses `brms` on the front-end so all the arguments we used previously apply here. Instead of the `brm()` function we use `ordbetareg()`.

```
# use ordbetareg to fit model
ord_fit_brms <- ordbetareg(Accuracy ~ Fluency_dummy,
  data = fluency_data,
  chains = 4,
  iter = 2000,
```

```
  backend = "cmdstanr",
  file = "model_beta_ordbeta"
)
```

Model parameters

Table 18 presents the model summary for our model.

Table 18

Model summary for ordered beta model

Parameter	Mean	95% CrI	pd
b_Intercept	-0.485	[-0.644, -0.325]	1
b_Fluency_dummy1	-0.239	[-0.468, -0.014]	0.981
phi	5.566	[4.568, 6.708]	1

Model parameters

Mean (μ)

Table 18 presents the model summary for our model. This summary looks just like the summary for our previous models, with `b_` representing the intercept and fluency contrast of μ. Similar to our above analyses, the CIs include zero and we can say there is no effect of `Fluency`.

Dispersion (ϕ)

Table 18 also includes an overall phi component. Similar to our other models we can model the variability as a function of fluency. Let's try this in our model:

```
m.phi <- ordbetareg(bf(Accuracy ~ Fluency_dummy,
                       phi ~ Fluency_dummy),
                    data=fluency_data,
                    backend = "cmdstanr",
```

```
                      file = "model_beta_ordbeta_phi",
                     iter = 2000,
                     cores=4,
                     phi_reg='both') # log phi
summary(m.phi)
```

Note the addition of the `phi_reg` argument. This argument allows us to include a model that explicitly models the dispersion parameter. Because I am modeling ϕ as a function of fluency, I set the argument to `both`.

Table 19

Model summary ordered beta regression with dispersion

Parameter	Mean	95% CrI	pd
b_Intercept	-0.489	[-0.651, -0.328]	1
b_phi_Intercept	1.759	[1.474, 2.036]	1
b_Fluency_dummy1	-0.229	[-0.456, -0.006]	0.978
b_phi_Fluency_dummy1	-0.109	[-0.509, 0.284]	0.707

In Table 19, `b_phi_Fluency_dummy1` is close enough to 0 relative to its uncertainty, we can say that in this case there likely aren't major differences in variance between the fluent disfluent conditions

Cutpoints

The model cutpoints are not reported by default, but we can access them with the R package `posterior` and the functions `as_draws` and `summary_draws`.

In Table 20, `cutzero` is the first cutpoint (the difference between 0 and continuous values) and `cutone` is the second cutpoint (the difference between the continuous values and 1). These cutpoints are on the logit scale and as such the numbers do not have a simple substantive meaning.

Table 20

Cutzero and cutone parameter summary

variable	mean	sd	mad	q5	q95
cutzero	-3.06	0.27	0.26	-3.51	-2.64
cutone	1.98	0.11	0.11	1.81	2.18

In general, as the cutpoints increase in absolute value (away from zero), then the discrete/boundary observations are more distinct from the continuous values. This will happen if there is a clear gap or bunching in the outcome around the bounds. This type of empirical feature of the distribution may be useful to scholars if they want to study differences in how people perceive the ends of the scale versus the middle.

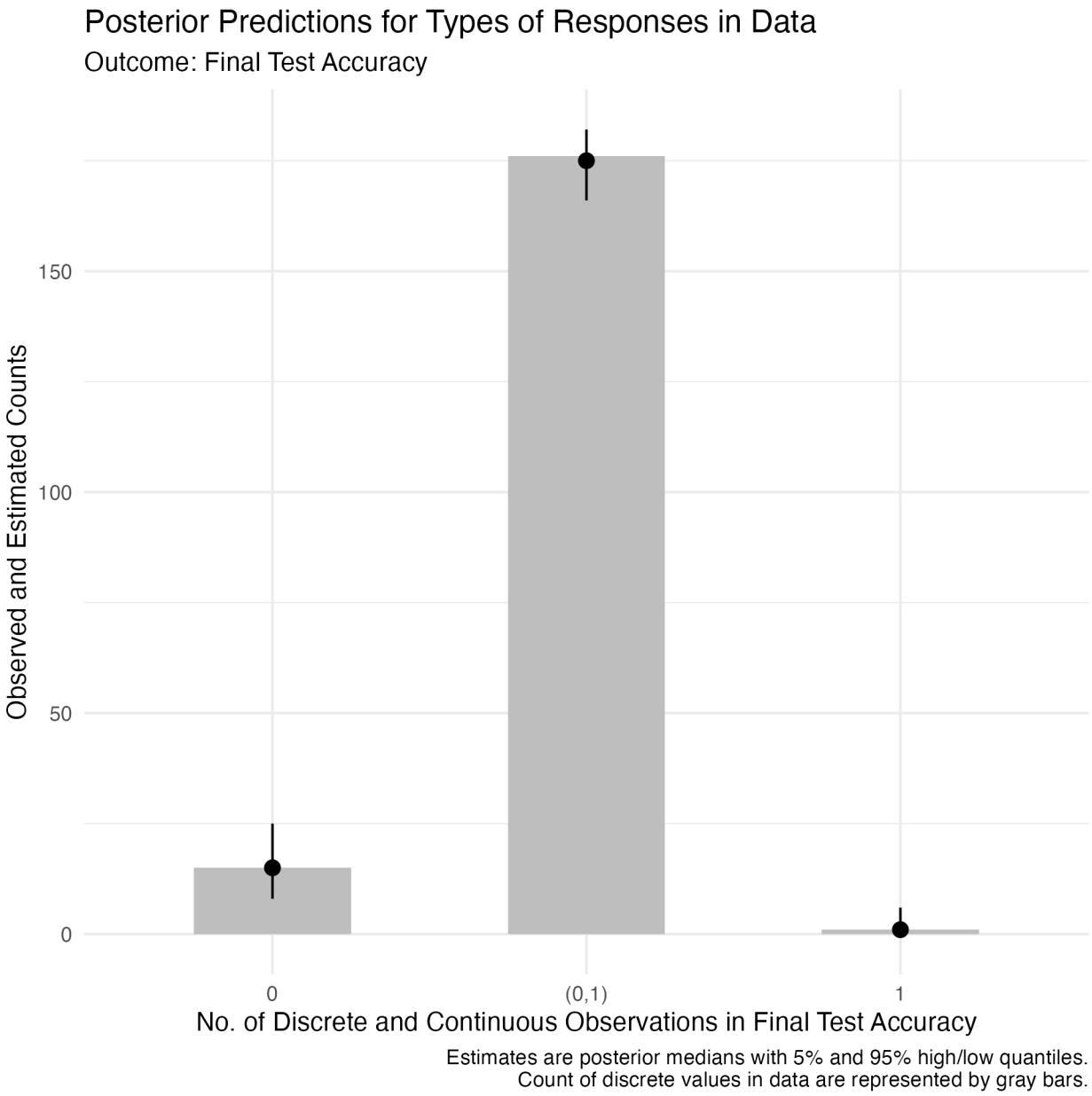
Ordered Beta regression model fit

The best way to visualize model fit is to plot the full predictive distribution relative to the original outcome. Because ordered beta regression is a mixed discrete/continuous model, a separate plotting function, `pp_check_ordbetareg`, is included in the `ordbetareg` package that accurately handles the unique features of this distribution. The default plot in `brms` will collapse these two features of the outcome together, which will make the fit look worse than it actually is. The `ordbetareg` function returns a list with two plots, `discrete` and `continuous`, which can either be printed and plotted or further modified as `ggplot2` objects. This can be observed in

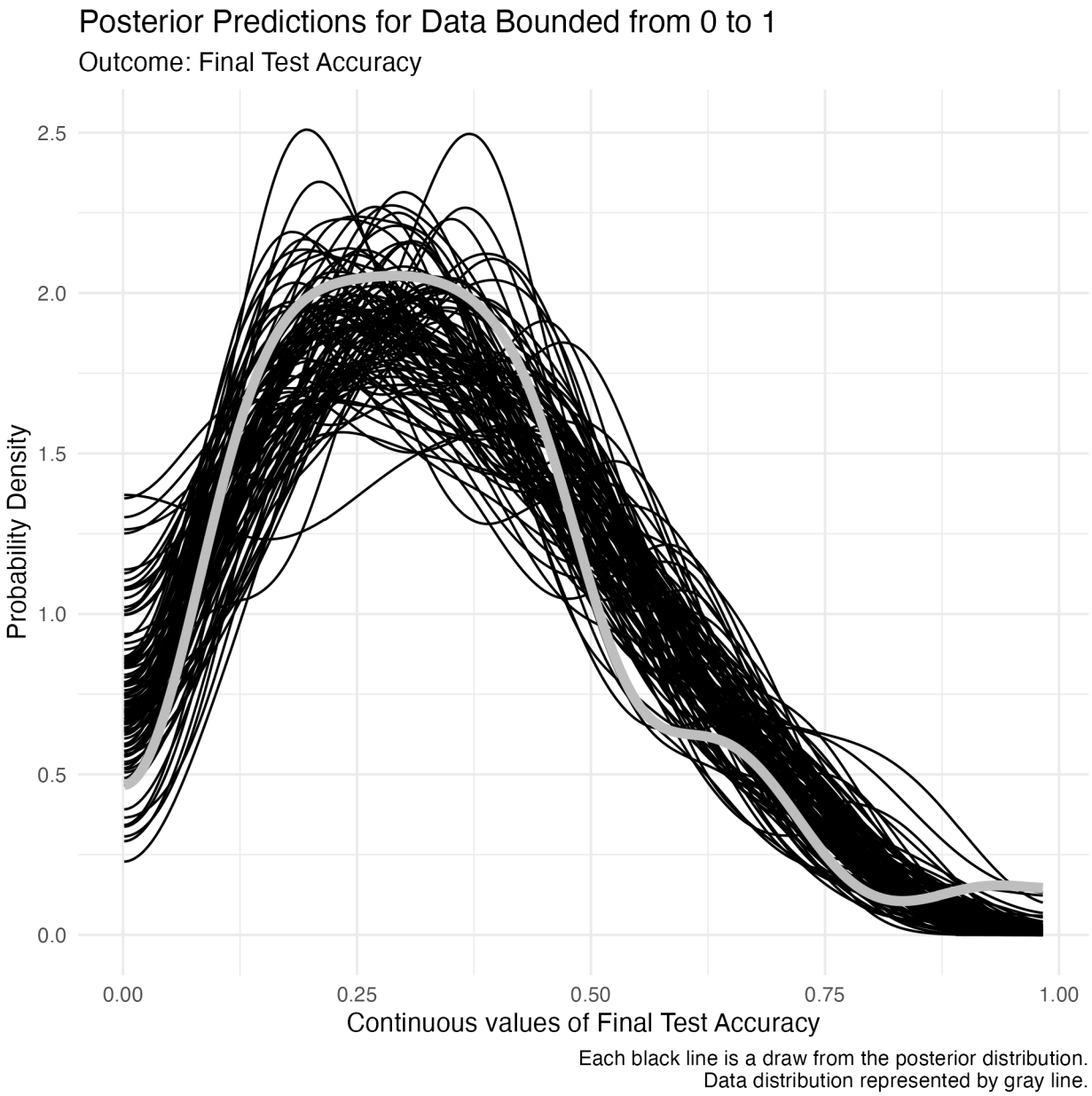
```
plots <- pp_check_ordbeta(ord_fit_brms,
  ndraws = 100,
  outcome_label = "Final Test Accuracy"
)


plots
```

```
$discrete
```


Posterior Predictions for Types of Responses in Data
Outcome: Final Test Accuracy
Observed and Estimated Counts
150
100
50
0
0
(0,1)
1
No. of Discrete and Continuous Observations in Final Test Accuracy
Estimates are posterior medians with 5% and 95% high/low quantiles.
Count of discrete values in data are represented by gray bars.

```
$continuous
```



The discrete plot which is a bar graph, shows that the posterior distribution accurately captures the number of different types of responses (discrete or continuous) in the data.

For the continuous plot shown as a density plot with one line per posterior draw, the model does a very good job at capturing the distribution.

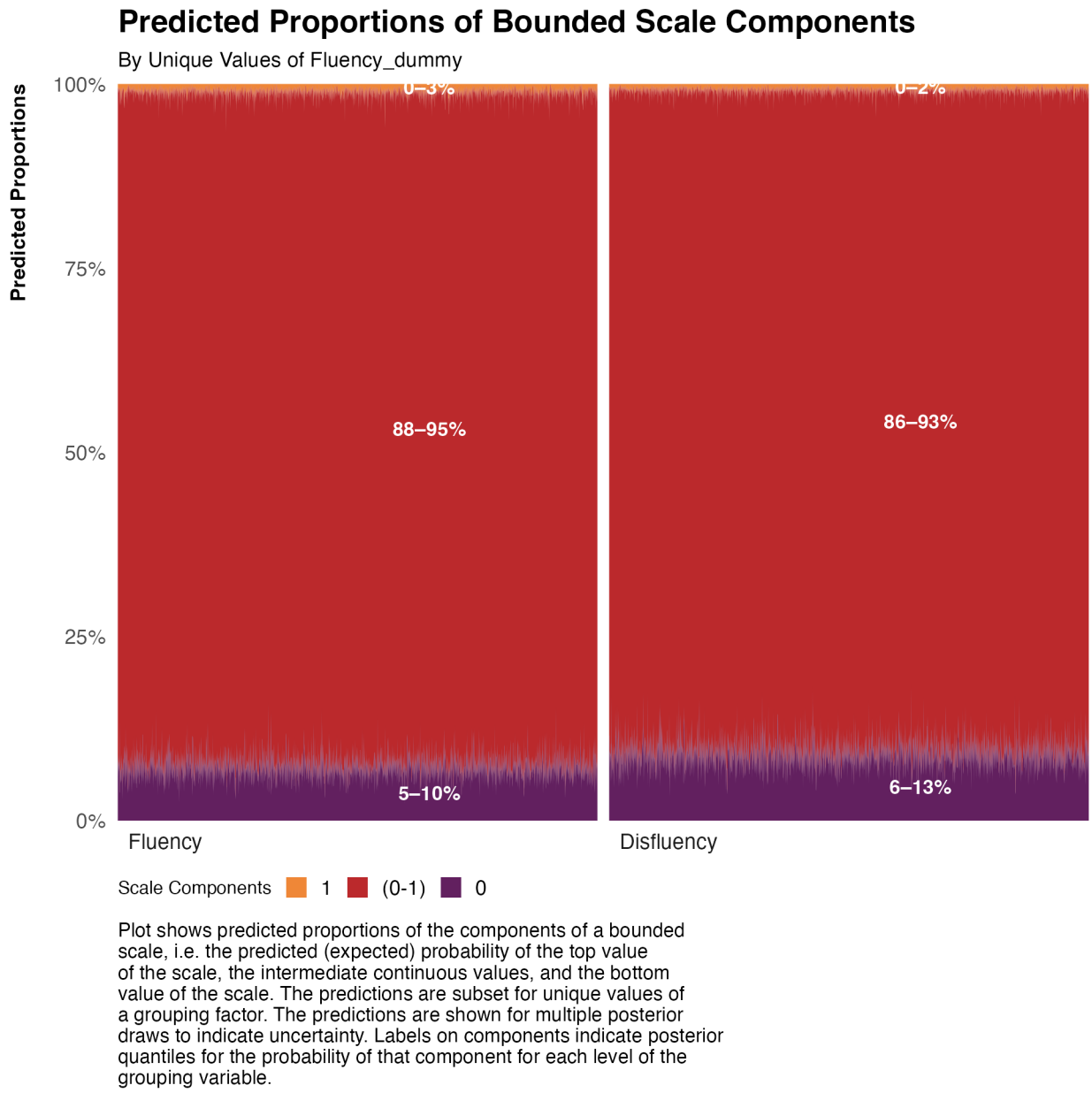
Overall, it is clear from the posterior distribution plot that the ordered beta model fits the data well. To fully understand model fit, both of these plots need to be inspected as they are conceptually distinct.

Model visualization. `ordbetareg` provides a visualization function called `plot_hess`. This function produces a plot of predicted proportions across the range of our `Fluency` factor. In Figure 11 we get predicted proportions for Fluency across the bounded scale. Looking at the figure we can see there is much overlap between Fluent and Disfluent instructors in the middle portion (μ) . However, we do see some small differences at the zero bounds.

```
plot_heiss(ord_fit_brms, grouping_fac="Fluency_dummy", recode_group_labels = c("Fluency"
```

Figure 11

Heiss Plot of Predicted Probablities across the scale (0-100)



Ordred Beta scale

In the `ordbetareg` function there is a `true_bound` argument. In the case where you data in not bounded between 0-1, you can use the argument to specify the bounds of the argument to fit the ordered beta regression. For example, you data might be bounded between 1 and 12. If so, you

can model it as such.

Discussion

The use of Beta regression in psychology, and the social sciences in general, is rare. With this tutorial, we hope to turn the tides. Beta regression models are an attractive alternative to models that impose unrealistic assumptions like normality, linearity, homoscedasticity, and unbounded data. Beyond these models, there are a diverse array of different models that can be used depending on your outcome of interest.

Throughout this tutorial our main aim was to help guide researchers in running analyses with proportional or percentage outcomes using Beta regression and some of it's alternatives. In the current example, we used real data from Wilford et al. (2020) and discussed how to fit these models in R, interpret model parameters, extract predicted probabilities and marginal effects, and visualize the results.

Comparing our analysis with that conducted by Wilford et al. (2020) we demonstrated that using the traditional approach (i.e., *t*-test) to analyze accuracy data can lead to inaccurate inferences. While we did reproduce the results from one particular analysis of Wilford et al. (2020), our use of Beta regression and its extensions suggest some nuance to their findings. With a traditional Beta regression model, which accounts for both the mean and precision/dispersion, revealed no effect of instructor fluency on performance. However, when using a zero-inflated Beta regression, which models the structural 0s in the data, we did find an effect of fluency. This effect was on the structural zero part, or inflated zero part, and not on the mean component. This finding was similar for the ZOIB model as well. Additionally, we fit an ordered Beta regression model (Kubinec, 2022), which models the process inclusive of zeros and ones. We did not observe a reliable finding for fluency on the mean portion of the model.

These analyses emphasize the importance of fitting a model that aligns with the nature of the data. The simplest and recommended approach when dealing with data that contains zeros and/or ones is to fit an ordered beta model, assuming the process is truly continuous. However, if you believe the process is distinct in nature, a ZIB or ZOIB model might be a better choice.

Ultimately, this decision should be guided by theory.

For instance, if we believe fluency influences the structural zero part of the model, we might want to model this process separately using a ZIB or ZOIB. With the current dataset, fluency might affect specific aspects of performance (such as the likelihood of complete failure) rather than general performance levels. This effect could be due to participant disengagement during the disfluent lecture. If students fail to pay attention because of features of disfluency, they may miss relevant information, leading to a floor effect at the test. If this is the case, we would want to model this appropriately. However, if we believe fluency effects general performance levels, a model that takes in to account the entire process accounting for the zeros and ones might be appropriate.

In the discussion section of Wilford et al. (2020), they were unable to offer a tenable explanation for performance differences based on instructor fluency. A model that accounts for the excess zeros in the dataset provides one testable explanation: watching a disfluent lecture may lead to lapses in attention, resulting in poorer performance in that group. These lapses, in turn, contribute to the observed differences in the fluent condition. This modeling approach opens a promising avenue for future research—one that would have remained inaccessible otherwise.

Not everyone will be eager to implement the techniques discussed herein. In such cases, the question becomes: what is the least problematic approach to handling proportional data? One option is to fit multiple models tailored to your specific use case. For example, if your data contains zeros, you could fit two models: a traditional OLS analysis without the zeros, and a logistic model to account for the zero/non-zero distinction. If your data contains both ones and zeros, you could fit separate models for the zeros and ones in addition to the OLS model. There are many defensible options to choose from. One thing we cannot recommend is transforming the values of your data in order to fit traditional statistical models.

In this tutorial, we demonstrated how to analyze these models from a Bayesian perspective. While we recognize that not everyone identifies as a Bayesian, implementing these models using a Bayesian framework is relatively straightforward—it requires only a single

package, lowering the barrier to entry. For those who prefer frequentist analyses, several R packages are available. For standard beta regression, the `betareg` package [@betareg] is a solid option, while more complex models such as zero-inflated and ordered beta regressions can be implemented using `glmmTMB` [@glmmTMB]. For fitting zero-one models, there is a new implementation in Cribari-Neto and Zeileis (2010), that allows you to model these types of data.

Conclusion

Overall, this tutorial emphasizes the importance of modeling the data you have. Although the example provided is relatively simple (a one-factor model with two levels), we hope it demonstrates that even with a basic dataset, there is much nuance in interpretation and inference. Properly modeling your data can lead to deeper insights, far beyond what traditional measures might offer. With the tools introduced in this tutorial, researchers now have the means to analyze their data effectively, uncover patterns, make accurate predictions, and support their findings with robust statistical evidence. By applying these modeling techniques, researchers can improve the validity and reliability of their studies, ultimately leading to more informed decisions and advancements in their respective fields.

References

Alves, J. C. R., Palma, G. R., & Lara, I. A. R. de. (2024). *Beta regression mixed model applied to sensory analysis*. https://arxiv.org/abs/2408.03240

Bartlett, M. S. (1936). The Square Root Transformation in Analysis of Variance. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *3*(1), 68–78. https://doi.org/10.2307/2983678

Bendixen, T., & Purzycki, B. G. (2023). Cognitive and cultural models in psychological science: A tutorial on modeling free-list data as a dependent variable in Bayesian regression. *Psychological Methods*. https://doi.org/10.1037/met0000553

Bürkner, P.-C. (2017). *{Brms}: An {r} package for {bayesian} multilevel models using {stan}*. *80*. https://doi.org/10.18637/jss.v080.i01

Bürkner, P.-C., & Vuorre, M. (2019). Ordinal Regression Models in Psychology: A Tutorial.

Advances in Methods and Practices in Psychological Science, *2*(1), 77–101. https://doi.org/10.1177/2515245918823199

Carpenter, S. K., Wilford, M. M., Kornell, N., & Mullaney, K. M. (2013). Appearances can be deceiving: instructor fluency increases perceptions of learning without increasing actual learning. *Psychonomic Bulletin & Review*, *20*(6), 1350–1356. https://doi.org/10.3758/s13423-013-0442-z

Cribari-Neto, F., & Zeileis, A. (2010). *Beta regression in {r}*. *34*. https://doi.org/10.18637/jss.v034.i02

Dolstra, E., & contributors, T. N. (2006). *Nix* [Computer software]. https://nixos.org/

Ferrari, S., & Cribari-Neto, F. (2004). Beta Regression for Modelling Rates and Proportions. *Journal of Applied Statistics*, *31*(7), 799–815. https://doi.org/10.1080/0266476042000214501

Fullerton, A. S., & Anderson, K. F. (2021). Ordered Regression Models: a Tutorial. *Prevention Science*, *24*(3), 431–443. https://doi.org/10.1007/s11121-021-01302-y

Gabry, J., Češnovar, R., Johnson, A., & Bronder, S. (2024). *Cmdstanr: R interface to 'CmdStan'*. https://mc-stan.org/cmdstanr/

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian data analysis* (Third). CRC. https://stat.columbia.edu/~gelman/book/

Heiss, A. (2021). *A guide to modeling proportions with bayesian beta and zero-inflated beta regression models*. http://dx.doi.org/10.59350/7p1a4-0tw75

Johnson, A., Ott, M., & Dogucu, M. (n.d.). *Bayes rules!: An introduction to applied bayesian modeling*. Routledge & CRC Press.

Kong, E. J., & Edwards, J. (2016). Individual differences in categorical perception of speech: Cue weighting and executive function. *Journal of Phonetics*, *59*, 40–57. https://doi.org/10.1016/j.wocn.2016.08.006

Kubinec, R. (2022). Ordered Beta Regression: A Parsimonious, Well-Fitting Model for Continuous Data with Lower and Upper Bounds. *Political Analysis*, *31*(4), 519–536. https://doi.org/10.1017/pan.2022.20

Kubinec, R. (2023). *Ordbetareg: Ordered beta regression models with 'brms'*. https://CRAN.R-project.org/package=ordbetareg

Lüdecke, D. (2018). *Ggeffects: Tidy data frames of marginal effects from regression models. 3*, 772. https://doi.org/10.21105/joss.00772

Lüdecke, D., Ben-Shachar, M. S., Patil, I., Wiernik, B. M., Bacher, E., Thériault, R., & Makowski, D. (2022). *Easystats: Framework for easy statistical modeling, visualization, and reporting*. https://easystats.github.io/easystats/

Makowski, D., Ben-Shachar, M. S., Chen, S. H. A., & Lüdecke, D. (2019). Indices of effect existence and significance in the bayesian framework. *Frontiers in Psychology*, *10*. https://doi.org/10.3389/fpsyg.2019.02767

Makowski, D., Ben-Shachar, M., & Lüdecke, D. (2019). bayestestR: Describing effects and their uncertainty, existence and significance within the bayesian framework. *Journal of Open Source Software*, *4*(40), 1541. https://doi.org/10.21105/joss.01541

Martin, K., Cornero, F. M., Clayton, N. S., Adam, O., Obin, N., & Dufour, V. (2024). Vocal complexity in a socially complex corvid: Gradation, diversity and lack of common call repertoire in male rooks. *Royal Society Open Science*, *11*(1), 231713. https://doi.org/10.1098/rsos.231713

McElreath, R. (2020). *Statistical rethinking: A bayesian course with examples in r and STAN* (2nd ed.). Chapman; Hall/CRC. https://doi.org/10.1201/9780429029608

Nelder, J. A., & Wedderburn, R. W. M. (1972). Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, *135*(3), 370–384. https://doi.org/10.2307/2344614

Nouvian, M., Foster, J. J., & Weidenmüller, A. (2023). Glyphosate impairs aversive learning in bumblebees. *Science of The Total Environment*, *898*, 165527. https://www.sciencedirect.com/science/article/pii/S0048969723041505

Paolino, P. (2001). Maximum Likelihood Estimation of Models with Beta-Distributed Dependent Variables. *Political Analysis*, *9*(4), 325–346. https://doi.org/10.1093/oxfordjournals.pan.a004873

R Core Team. (2024). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Rodrigues, B., & Baumann, P. (2025). *Rix: Reproducible data science environments with 'nix'*. https://docs.ropensci.org/rix/

Shrestha, S., Sigdel, K., Pokharel, M., & Columbus, S. (2024). Big five traits predict between- and within-person variation in loneliness. *European Journal of Personality*, 08902070241239834. https://doi.org/10.1177/08902070241239834

Sladekova, M., & Field, A. P. (2024). *In search of unicorns: Assessing statistical assumptions in real psychology datasets*. https://doi.org/10.31234/osf.io/4rznt

Smith, K. E., Panlilio, L. V., Feldman, J. D., Grundmann, O., Dunn, K. E., McCurdy, C. R., Garcia-Romeu, A., & Epstein, D. H. (2024). Ecological momentary assessment of self-reported kratom use, effects, and motivations among US adults. *JAMA Network Open*, *7*(1), e2353401. https://doi.org/10.1001/jamanetworkopen.2023.53401

Smithson, M., & Verkuilen, J. (2006). A better lemon squeezer? Maximum-likelihood regression with beta-distributed dependent variables. *Psychological Methods*, *11*(1), 54–71. https://doi.org/10.1037/1082-989x.11.1.54

Team, S. D. (2023). *Stan: A probabilistic programming language*. https://mc-stan.org

Toftness, A. R., Carpenter, S. K., Geller, J., Lauber, S., Johnson, M., & Armstrong, P. I. (2017). Instructor fluency leads to higher confidence in learning, but not better learning. *Metacognition and Learning*, *13*(1), 1–14. https://doi.org/10.1007/s11409-017-9175-0

Vuorre, M. (2019, February 18). *How to Analyze Visual Analog (Slider) Scale Data?* https://vuorre.com/posts/2019-02-18-analyze-analog-scale-ratings-with-zero-one-inflated-beta-models

Wickham, H. (2017). *Tidyverse: Easily install and load the 'tidyverse'*. https://CRAN.R-project.org/package=tidyverse

Wilford, M. M., Kurpad, N., Platt, M., & Weinstein-Jones, Y. (2020). Lecturer fluency can impact students' judgments of learning and actual learning performance. *Applied Cognitive*

Psychology, *34*(6), 1444–1456. https://doi.org/10.1002/acp.3724

Wilkes, L. N., Barner, A. K., Keyes, A. A., Morton, D., Byrnes, J. E. K., & Dee, L. E. (2024). Quantifying co-extinctions and ecosystem service vulnerability in coastal ecosystems experiencing climate warming. *Global Change Biology*, *30*(7), e17422. https://doi.org/10.1111/gcb.17422

Witherby, A. E., & Carpenter, S. K. (2022). The impact of lecture fluency and technology fluency on students' online learning and evaluations of instructors. *Journal of Applied Research in Memory and Cognition*, *11*(4), 500–509. https://doi.org/10.1037/mac0000003